UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UNE ARCHITECTURE ALTERNATIVE À LA TECHNOLOGIE "MÉMOIRE ADRESSABLE PAR CONTENU"

PATRICK MAHONEY
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2006

Your file Votre référence
ISBN: 978-0-494-17955-0
Our file Notre référence
ISBN: 978-0-494-17955-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION D'UNE ARCHITECTURE ALTERNATIVE À LA TECHNOLOGIE "MÉMOIRE ADRESSABLE PAR CONTENU"

présenté par: MAHONEY Patrick
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

Mme NICOLESCU Gabriela, Doct., présidente
M. BOIS Guy, Ph.D., membre et directeur de recherche
M. SAVARIA Yvon, Ph.D., membre et codirecteur de recherche
M. DAVID Jean-Pierre, Ph.D., membre

REMERCIEMENTS

Je remercie mon directeur Guy Bois pour avoir grandement facilité mon retour à l'École Polytechnique. Il a su être disponible tout au long de cette aventure en me fournissant support et conseils.

Je remercie mon co-directeur Yvon Savaria grâce à qui j'ai pu découvrir le monde de la microélectronique tout en ayant le nécessaire pour payer le loyer et l'épicerie.

Je remercie Patrice Plante pour avoir pris le temps de répondre à mes nombreuses questions ainsi que pour m'avoir donné accès aux ressources de son employeur pendant la durée du projet.

Je remercie mes parents sans qui tout ceci n'aurait pas été possible.

RÉSUMÉ

La technologie *Content Addressable Memory* (CAM) est une technologie mémoire qui permet d'accéder aux données à partir de la donnée même plutôt qu'à partir de son adresse. Elle est utilisée par une grande portion des applications nécessitant un haut débit d'accès mémoire. La raison est simple: elle effectue ses opérations de recherche et d'insertion en un seul cycle.

La technologie CAM parvient à réaliser des recherches en un seul cycle comparant simultanément la donnée fournie par l'usager à toutes les données présentes en mémoire. Si la donnée recherchée est présente, l'information associée à la donnée recherchée est présentée à l'usager.

Elle possède toutefois plusieurs inconvénients majeurs. Tout d'abord, le parallélisme des comparaisons demande une grande quantité de comparateurs qui réalisent leur tâche en même temps. Ainsi, pour une capacité donnée, la technologie CAM occupe une grande quantité d'aire de silicium. Aussi, le fonctionnement simultané des comparateurs demande une grande quantité d'énergie à chaque opération de recherche. Les designers doivent donc tenir compte de la grande consommation de puissance et de la grande dissipation de chaleur qu'implique l'utilisation de la technologie CAM. Finalement, les fournisseurs des compilateurs de CAM exigent des frais de licence qui rendent la technologie inabordable pour plusieurs.

Les alternatives consistent principalement au recours à la technologie *Random Access Memory* (RAM) utilisée en combinaison avec des algorithmes de recherche et d'insertion conventionnels: structures arborescentes, hashing, etc. Les performances qu'elles permettent d'atteindre ne sont tout simplement pas les mêmes.

Le présent projet consiste en la conception d'une architecture mémoire offrant des performances similaires à celles des CAM tout en tirant profit des caractéristiques de la technologie RAM: faible coût, forte densité et faible consommation.

L'architecture propose de disposer un certain nombre d'unités de RAM en parallèle et de les utiliser comme table de hashing, chaque table ayant une unité réalisant une fonction de hashing et la combinaison des deux représentant une couche. Lors d'une insertion, chaque couche vérifie si la rangée à laquelle correspond la donnée à insérer est libre. L'insertion est faite dans la première table présentant une entrée disponible. Lors d'une recherche, chaque couche accède à la rangée de l'unité de RAM à laquelle correspond la donnée recherchée et compare la donnée accédée à celle fournie par l'usager. Une logique se charge de rassembler les résultats des comparaisons et d'acheminer l'information pertinente en sortie.

Cette architecture tire profit de la technologie RAM, mais ne peut garantir l'absence de tout échec lors des insertions. En effet, si les rangées de toutes les couches sont occupées lors d'une tentative insertion, celle-ci ne peut être réalisée. Le présent projet analyse donc les performances de l'architecture et tente d'évaluer son coût. Tout d'abord, le comportement de l'architecture a été modélisé par un simulateur et les premiers indices de performances ont ainsi pu être obtenus. Un peu comme il était permis de croire, les simulations nous ont confirmé que pour un nombre d'insertions donné, une augmentation du nombre de couches ou une augmentation de la capacité totale de l'architecture permettent d'obtenir un plus petit nombre d'échecs. Ces constatations sont toutefois basées sur la présence de fonctions de hashing idéales. Ces fonctions ont la caractéristique de briser toute corrélation dans les données à insérer pour ainsi disperser de façon uniforme les adresses mémoire générées.

Toutefois, les simulations ne nous permettent que difficilement d'identifier les configurations qui mènent à des taux d'échecs inférieurs à 10^{-4}. Pour combler ce manque, un modèle analytique a donc été conçu. Celui-ci nous permet d'identifier les configurations menant à des taux d'échecs de n'importe quel ordre de grandeur. Il permet ainsi de définir de façon précise les performances de l'architecture. Il est important de noter que ce modèle se base aussi sur l'hypothèse de fonctions de

hashing idéales.

Pour évaluer le coût de l'architecture, une réalisation VLSI a été effectuée. Les métriques obtenues se comparent avantageusement aux fiches techniques des modules CAM. Elles nous permettent d'estimer que pour une capacité donnée, le rapport d'aire de l'architecture proposée par rapport à une cellule de CAM est de 2.4. En ce qui concerne la consommation, ce rapport est de 4.7.

Ces résultats nous indiquent que pour reproduire le comportement d'une unité de CAM, il est possible de recourir à une architecture de hashing parallèle possédant une capacité supérieure à cette unité sans nécessairement avoir à débourser pour une plus grande consommation d'aire de silicium, de consommation de puissance ou encore de dissipation de chaleur.

ABSTRACT

Content Addressable Memories (CAMs) allow lookup operations to be executed in a single cycle. This feature enables many applications to sustain high data throughput in a deterministic manner. It does so by comparing the value to look for with every item in memory. On a successful lookup, information associated with the sought after data is gated to the output.

These parallel comparisons cause CAM units to have high area and power consumption metrics. Another key liability of CAM technology resides in the high licensing fees imposed by CAM vendors.

These factors have designers looking for alternatives. Most of them consist in using Random Access Memory (RAM) technology combined with classic algorithms such as hashing or tree-based algorithms. The performances of the alternatives do not match the performances of CAMs.

The current project aims at offering a RAM based alternative offering performances similar to ones of CAMs while leveraging on RAM technology to offer improved area and power consumption metrics.

The proposed architecture is structured as sets of modules implementing multiple processing layers, each consisting of a hashing function, a static RAM unit and some combinational logic. The hashing functions translate the incoming data into an address used by the RAM cell. When an insertion attempt is made, all RAM units verify the availability of the row associated with the data to insert. The insertion is made on the highest layer among those signaling an empty row. On lookup operations, all layers access the row associated with the sought after data and compare it with the accessed value. Glue logic takes care of regrouping the results and gating the relevant information to the output.

The proposed architecture offers an interesting cost saving alternative to CAMs, yet it cannot guarantee to successfully realize every insertion attempt. Attempts

will fail if corresponding rows in all layers are occupied. The current project aims at analyzing and quantifying this performance behaviour so as to provide designers all the tools they need to fully benefit from this architecture.

A simulator was first used to obtain the first performances metrics. As we had guessed it, for a given number of insertions attempts, performances are enhanced by increasing the number of layers or by increasing the total architecture capacity. However, it is important to note that the simulator uses high quality hashing functions in its architecture model. These hashing functions possess the characteristic of breaking any correlation in search key traffic so as to yield uniformly dispersed hash keys. Another drawback of the simulator was to severely limit the analysis of the performance domain where collision rates get lower than 10^{-4}. Prohibitive simulation times make it very difficult to identify architecture configurations yielding to lower failure rates.

In order to fill this gap, an analytical model of the architecture was created. This model assumes perfect hashing functions, yet it covers all of the performance domain. The analytical model allows to identify configurations yielding to failure rates of any magnitude.

In order to evaluate the cost of this architecture, a VLSI implementation was realized. It allowed us to make certain comparisons with commercial CAMs. Among them, it was found that the parallel hashing architecture consumed less die area than CAM units by a factor of 2.4. The value reaches 4.7 when comparing power consumption metrics.

In order to emulate a CAM unit of a given size, it is thus possible to use an oversized unit of parallel hashing memories while still offering improved die area and power consumption metrics.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

LISTE DES NOTATIONS ET DES SYMBOLES

CAM . . . . . Content Addressable Memory

RAM . . . . . Random Access Memory

CRC . . . . . Cyclic Redundancy Check

VLSI . . . . . Very Large Scale Intergrated Circuit

PDT . . . . . Pulled Down Transistor

α . . . . . Taux de charge de l'architecture

L . . . . . Nombre de couches de l'architecture

AVANT PROPOS

Le présent document prend la forme d'un mémoire par articles, tel que défini par le Bureau des Affaires Académiques (BAA) de l'École Polytechnique de Montréal. Les règlements du BAA stipulent que seuls les articles de revue permettent la rédaction d'un mémoire par article. Comme de fait, l'annexe de ce mémoire en contient un en annexe. Cependant, l'auteur de ce mémoire a cru pertinent d'y insérer en plus un article de conférence. Cet article brosse un portrait des étapes initiales du projet qui ont mené au travaux présentés dans l'article de revue.

L'information que contient le corps du mémoire a pour but de synthétiser, de complémenter et de supporter les deux publications écrites par le groupe de recherche dont fait parti l'auteur de ce mémoire.

Le lecteur saura trouver dans le corps du mémoire une synthèse globale des travaux effectués dans le cadre du projet. La lecture des articles n'est pas essentielle à la compréhension d'ensemble. Cependant, elle permettra aux lecteurs curieux de saisir l'ensemble des tâches accomplies par l'équipe de recherche.

INTRODUCTION

Les dix dernières années ont donné lieu à une explosion de la demande générale en services réseau de toutes sortes. L'intégration de services comme la téléphonie cellulaire, la messagerie SMS, les Blackberries et le *podcasting* dans le quotidien d'une tranche importante de la population est en partie responsable de ce phénomène. Les conséquences de ces généralisation et décentralisation de la demande en services réseau sont nombreuses. Du point de vue des fournisseurs, les deux plus importantes sont l'amélioration des services et la baisse des coûts.

Ces transformations à la demande se reflètent directement chez les fabricants d'équipements réseau et ce, à tous les niveaux de l'échelle de production. Les fournisseurs de solutions intégrées tout comme les fabricants de puces doivent se soumettre aux nouvelles contraintes. Les produits offerts doivent intégrer un plus grand nombre de solutions et être offerts à un coût moindre.

L'explosion du nombre d'applications réseau a entraîné un accroissement important de la quantité d'information transférée à l'échelle mondiale. Les applications de traitement de paquets ont ainsi connu une grande effervescence, elles aussi. Plusieurs de ces applications ont un point commun, l'avantage que leur procure un recours à la technologie des "Content Addressable Memories", ou CAM.

Description la Technologie CAM

La technologie mémoire CAM se prête très bien à plusieurs applications de traitement de paquets, puisqu'elle permet d'effectuer les opérations de recherche et d'insertion en un seul cycle. Elle permet ainsi de soutenir des débits très élevés.

Les rangées des unités de CAM contiennent typiquement au moins deux informations essentielles: l'étiquette et la donnée utile. Sur la première sont effectuées les comparaisons avec les données recherchées. La seconde représente la donnée que cherche à obtenir l'usager. Par exemple, dans une application d'aiguillage de paquets, les adresses seraient contenues dans la portion étiquette, tandis que les identificateurs des ports permettant de rejoindre les adresses seraient présents dans la portion donnée utile.

Lors des opérations de recherche, l'élément recherché est comparé à la portion étiquette de toutes les entrées en mémoire. Si une comparaison s'avère réussie, la portion donnée utile de cette rangée est acheminée vers la sortie. La Fig.1 illustre le fonctionnement interne d'une CAM.

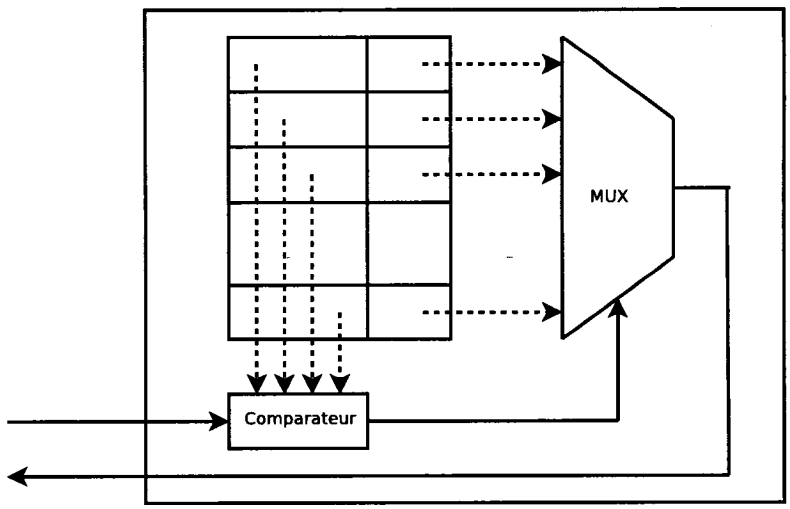

Figure 1 Fonctionnement interne de la CAM

Ces comparaisons parallèles entraînent toutefois une grande consommation de puissance, ainsi qu'une forte dissipation de chaleur. En plus de ces inconvénients, les frais non-récurrents très élevés exigés par les fournisseurs rendent l'accès à la technologie très difficile. Finalement, la superficie de silicium qu'occupent les cellules

de CAM représente un autre aspect du coût très élevé associé à l'utilisation de cette technologie. La table II.1 compare quelques métriques de la CAM avec celles de la technologie RAM.

Tableau 1 Comparaison de métriques d'une cellule de RAM statique avec celles d'une CAM binaire. Technologie CMOS $0.18\mu m$. Fréquence d'opération 100 MHz.

	CAM	RAM	Facteur
Coût $(mm^2/Mbit)$	19.2	6.75	2.8
Consommation Opération de lecture $(mW/Mbit)$	410	80.9	5.1
Consommation Opération d'écriture $(mW/Mbit)$	735	89.3	8.23

Le présent projet propose une architecture alternative à la technologie CAM. Elle a recours à la technologie RAM pour pouvoir pallier aux inconvénients mentionnés ci-haut. Elle a recours au *hashing* pour parvenir à offrir une fonctionnalité similaire à celle de la technologie CAM.

Hashing

Le hashing est un algorithme qui permet d'effectuer des recherches et des insertions de données en un seul accès mémoire. Une donnée à insérer en mémoire (la clé de recherche) est associée à une adresse de cette mémoire (la clé de hashing) à l'aide d'une logique déterministe (la fonction de hashing). Lors d'une opération de recherche, ce n'est qu'à cette adresse que la mémoire doit être accédée. Pour les insertions, ce n'est qu'à cette adresse qu'il est permis d'insérer la donnée. L'algorithme de hashing, qui tente de produire un patron d'écriture aussi aléatoire que possible, offre donc des performances intéressantes, mais il est nécessaire de

gérer les collisions qui en résultent. Celles-ci surviennent lorsqu'une insertion doit être faite à une adresse qui est déjà occupée par une autre donnée. La figure 2 illustre les différents participants de l'algorithme de hashing.

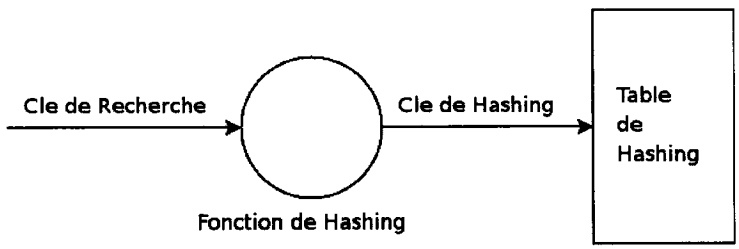

Figure 2 Les acteurs d'une architecture de hashing.

L'architecture présentée dans ce document offre une alternative économique à la technologie CAM. En effet, en ayant recours à l'algorithme de hashing et tirant profit des caractéristiques de la technologie RAM, l'architecture présentée parvient à émuler le comportement des CAM tout en offrant des métriques d'aire et de consommation avantageuses.

Présentation du Mémoire

Le mémoire contient trois sections en plus des annexes. Tout d'abord, le second chapitre consiste en une revue de littérature portant sur la technologie CAM. Le troisième chapitre, quant à lui, présente au lecteur l'information nécessaire à la pleine compréhension des articles. Il décrit les différentes décisions qui ont défini les bases du projet de recherche. De plus, il résume le contenu des deux articles insérés en annexe pour fournir au lecteur une vue d'ensemble lui permettant de saisir les principaux aspects techniques en jeu.

Le quatrième chapitre traite de la réalisation VLSI. Il aborde tout d'abord le design de la réalisation ainsi que la méthodologie employée, pour finalement présenter

les performances résultantes en les comparant à celles de la CAM. Finalement, les annexes contiennent les deux articles écrits par l'équipe impliquée dans cette recherche.

CHAPITRE 1

REVUE DE LITTÉRATURE

Ce chapitre présente un survol de l'état de la recherche dans le domaine de la technologie CAM en 2006. Sa lecture permettra au lecteur de situer l'apport exact du projet par rapport à ce qui a déjà été fait.

Comme le lecteur pourra le constater, plusieurs travaux tentent de pallier aux lacunes de la technologie CAM. Plusieurs proposent des modifications à la cellule de base de la CAM permettant une réduction de sa consommation, souvent au détriment d'une autre métrique comme le délai de l'opération ou encore l'aire de silicium. D'autres proposent des solutions logicielles ou architecturales. Les différentes solutions seront tout d'abord décrites. Ensuite, le présent projet sera situé par rapport aux autres travaux, permettant ainsi au lecteur d'identifier sa contribution exacte.

1.1 Modifications à la cellule de base de la CAM

Une grande portion des efforts investis pour pallier au problème de consommation de la technologie CAM est centrée sur la cellule de base mémorisant un bit de CAM.

La Fig. 1.1 illustre une cellule de base typique d'une unité CAM. Elle est composée d'une cellule SRAM jumelée à une porte XOR dont la sortie contrôle une résistance de rappel vers le niveau bas, le PDT (*pulled down transistor*). Si un bit en entrée, représenté à la Fig. 1.1 par les valeurs complémentaires bit et /bit, n'est pas

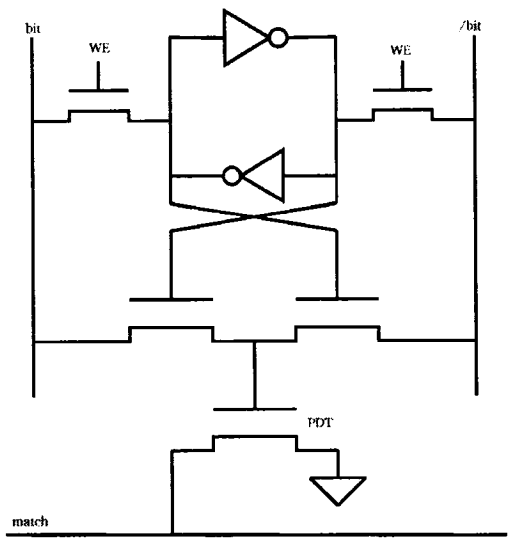

Figure 1.1 Cellule CAM de base

identique à la valeur en mémoire, le PDT sera activé et le signal *match* sera amené à la terre. Sinon, le signal *match* sera dans un état HZ (haute impédance). Le signal WE (write enable), si activé, permettra à la donnée d'être écrite à la cellule SRAM.

Une unité de CAM dispose les cellules en mosaïque en joignant les signaux *WE*, */bit*, *bit* et *match* des cellules voisines. La valeur par défaut du signal *match* est obtenue à l'aide d'un simple mécanisme de résistance de rappel vers le niveau haut. Le signal *match* d'une rangée donnée sera dans l'état HZ si et seulement si toutes les cellules de base de la rangée contiennent une valeur identique à leur bit d'entrée respectif. La matrice résultante est illustrée à la Fig. 1.2.

Le désavantage principal de la cellule décrite ci-haut se situe au niveau du déchargement fréquent du signal *match* au travers du PDT. En effet, le signal *match* se décharge pour toutes les rangées contenant une portion étiquette non identique à la donnée présente en entrée. Plusieurs projets ((Natarajan 2003),(Efthymiou, 2002), (Pa-

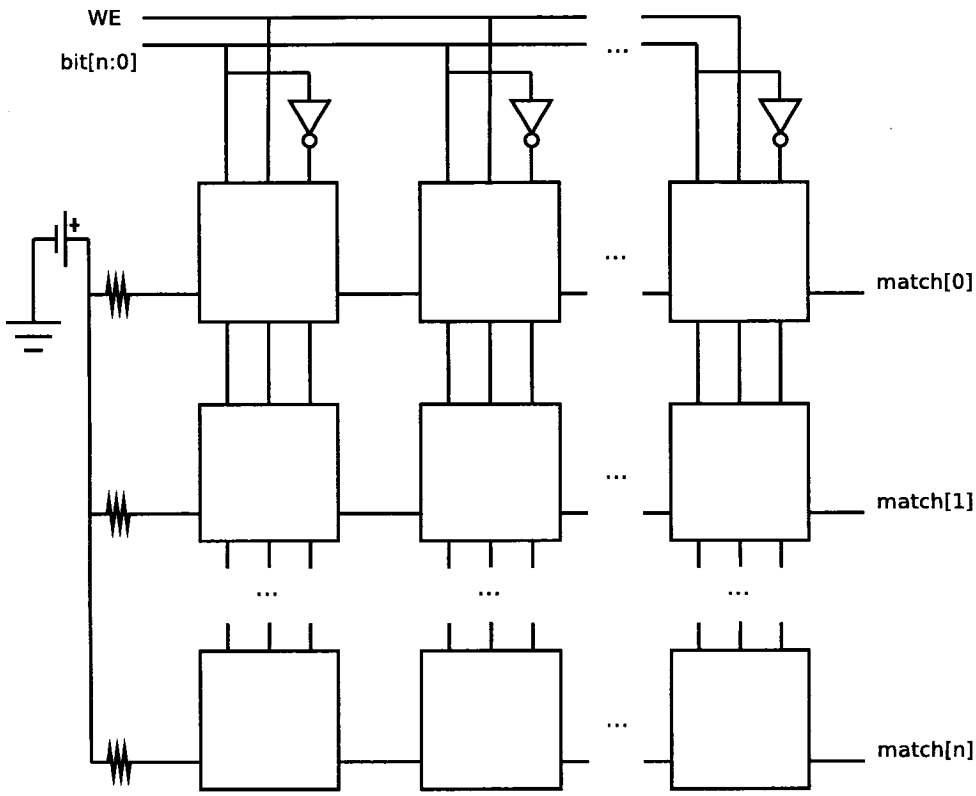

Figure 1.2 Mosaïque de cellules de CAM.

giamtzis 2004),(Delgado-Frias 2005),(Liu 2001)) ont entrepris de modifier la cellule de base de la CAM dans le but de minimiser sa consommation. Par exemple, Lin (Lin, 2003) a tenté de réduire au minimum le nombre de cellules avec un PDT actif. Il propose la cellule illustrée à la figure 1.3 dans laquelle les sorties complémentaires de la cellule SRAM sont inversées par rapport à celles de la figure 1.1. Ainsi, le transistor présent sur le signal *match* sera actif si le bit en entrée est identique à celui contenu en mémoire. Une configuration "NAND" est obtenue en liant à la terre le signal *match* à l'entrée de la première cellule et en plaçant une résistance de rappel vers le signal haut à la sortie. Le signal *match* obtenu sera ainsi à la terre si et seulement si la rangée contient un champ étiquette identique à la donnée

recherchée.

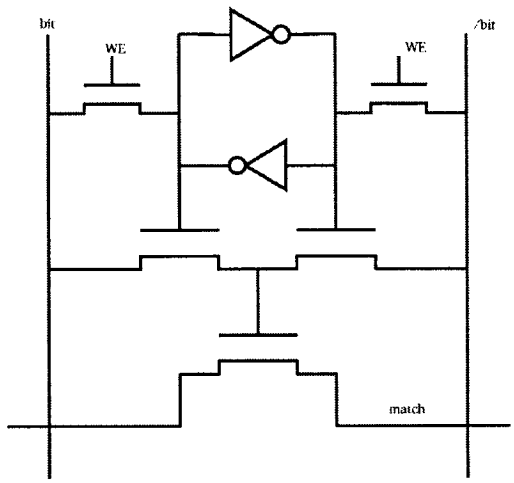

Figure 1.3 Cellule CAM de base proposée par Lin (Lin, 2003)

Dans cette configuration, seules les rangées contenant un mot identique à celui présenté en entrée posséderont un signal *match* déchargé à la terre. Il en résulte une réduction de la consommation au détriment d'un plus grand délai. En effet, en disposant les portes de façon sérielle, il en résulte un réseau de portes de transmission dont le délai est proportionnel au carré du nombre de portes (Savaria 1988). Cet inconvénient limite la largeur des rangées de CAM pour lesquelles la configuration sérielle représente une solution adéquate. Zukowski (Zukowski 1997) propose à cet effet une solution intermédiaire qui laisse le choix à l'usager de la configuration exacte.

La figure 1.4 présente une configuration qui permet au concepteur de limiter les effets de la sérialisation des portes sur le délai. En effet, il est libre de disposer k portes de façon sérielle et les autres, en parallèle. De cette façon, le signal *match* ne pourra être déchargé qu'en la présence d'une comparaison partielle réussie puisque le signal *match* ne sera préchargé que si les bits D_0 à D_{k-1} sont tous identiques à leur bit d'entrée respectifs. Le concepteur est ainsi libre de définir le meilleur

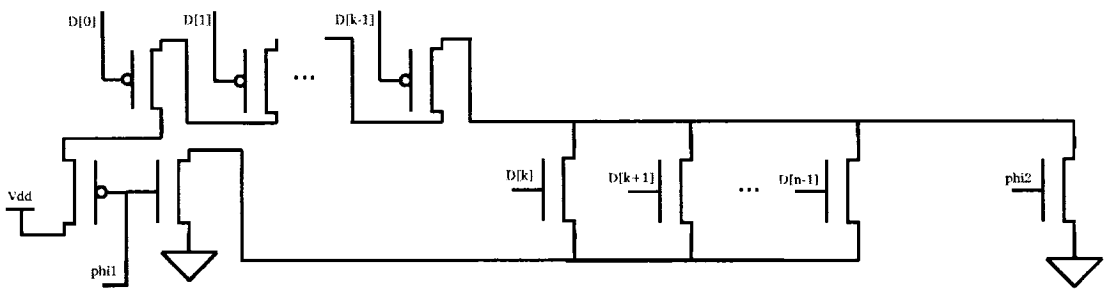

Figure 1.4 La précharge sélective est un compromis entre la latence et la consommation proposé par Zukowski (Zukowski 1997).

compromis entre la consommation et le délai.

1.2 Modifications Architecturales

L'intégration à un design d'une unité de CAM optimisée à l'aide d'une modification à le cellule de base n'est pas une tâche simple. En effet, il est tout d'abord nécessaire que cette cellule soit intégrée à un compilateur de CAM pour que celui-ci puisse générer l'unité de CAM optimisée. En ce sens, il est plus simple et surtout plus rapide d'intégrer à un design une unité de CAM optimisée à l'aide de modifications faites au niveau architectural.

Lin et Chang (Lin 2000) proposent une architecture en plusieurs points similaires à celle présentée dans cet ouvrage. Elle rajoute aux rangées de la CAM une clé de hashing, appelée "paramètre" par les auteurs, obtenue en appliquant une fonction de hashing choisie par le concepteur sur la donnée à conserver en mémoire. La figure 1.5 illustre cette architecture. Chaque rangée de la CAM est ainsi composée du paramètre et de la donnée à conserver. Les comparateurs sont conçus de façon à tout d'abord comparer le paramètre et ensuite n'effectuer la comparaison de la donnée que si la comparaison du paramètre est réussie. Puisque le paramètre est

choisi de façon à être beaucoup plus petit que la donnée, il en résulte une économie du nombre moyen de bits comparés à chaque accès.

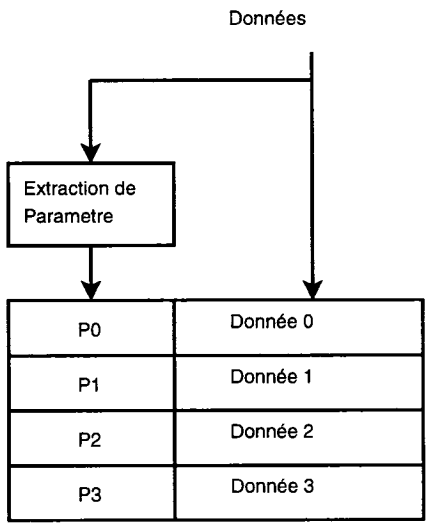

Figure 1.5 Lin et Chang (Lin 2000) proposent de comparer la donnée seulement si la comparaison du paramètre est réussie.

Certains chercheurs (Kartalopoulos 1998) suggèrent aussi d'utiliser des unités de RAM en mode "correspondance directe" pour émuler le comportement d'une CAM. Cette méthode consiste à associer une donnée à l'adresse égale à cette même donnée. Les avantages de cette méthode sont évidents, puisqu'elle permet d'obtenir des performances de la CAM avec des unités issues de la technologie RAM. Toutefois, un nombre limité d'applications se prêtent à ce genre d'architecture. En effet, la correspondance directe nécessite une entrée disponible en RAM pour chaque élément qu'il est possible de rencontrer durant l'exécution de l'application. Par exemple, si l'application nécessite l'entreposage en CAM d'adresses IPv4, dont la taille est de 32 bits, la cellule de RAM se devrait alors de posséder 2^{32} entrées. Le coût d'une telle cellule serait alors démesuré. Ce genre de design est donc réservé pour des applications possédant un domaine de données de taille restreinte. Par exemple, une application de traitement de paquets ATM, utilisant le champ de 12

bits VCI (*virtual circuit identifier* ou le champs de 16 bits VPI *virtual path identifier* comme clés de recherche, serait un bon candidat.

Certains chercheurs ont identifié le hashing (Sedgewick 2002) comme étant un outil très efficace pour pallier a ce problème. Cette technique consiste à utiliser la correspondance directe, mais avec une clé de hashing agissant comme indice, plutôt qu'avec la donnée à insérer. Il s'agit d'ailleurs d'un processus utilisé fréquemment dans des systèmes de cache utilisant la technologie RAM (Hennessy 2003). Le problème propre au hashing se situe au niveau des collisions. Dépendamment des besoins de l'application, il peut être acceptable ou non de rejeter certaines données à insérer.

Plusieurs méthodes sont utilisées pour gérer les collisions. Par exemple, Seznec (Seznec 1993) propose une architecture de hashing multiple dans le cadre d'une unité de mémoire cache. La figure 1.6 illustre le tout.

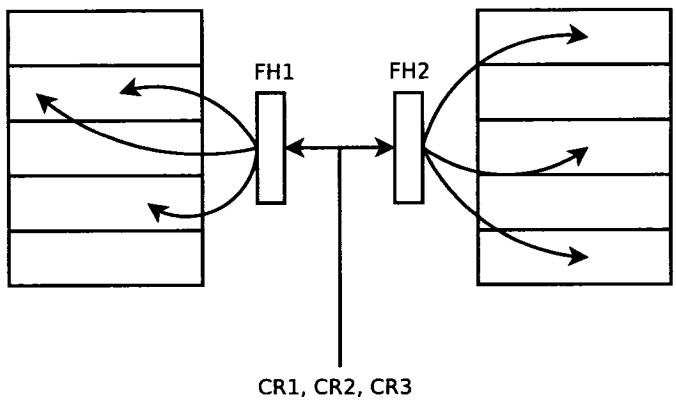

Figure 1.6 Seznec (Seznec 1993) propose un système de cache avec deux fonctions de hashing pour minimiser les collisions.

L'architecture illustrée à la figure 1.6 comporte deux bancs de cache, chacun étant composé d'une unité de RAM et d'un bloc réalisant la fonction de hashing. Lors d'une insertion, les bancs sont accédés selon un ordre de priorité. Ainsi, une inser-

tion ne sera faite dans le second banc que s'il y a collision dans le premier. De plus, s'il y a collision dans le second banc, une certaine logique se chargera de gérer la situation, rejetant une des trois données en jeu selon un algorithme prédéfini. La présence de deux fonctions de hashing distinctes permet à un groupe d'éléments qui entrent en collision dans un banc de cache d'avoir de fortes probabilités de pouvoir être tous contenus dans le second.

L'architecture des caches "skewed associative" n'est pas sans rappeler celles des caches associatifs par ensembles. La principale différence se situe au niveau du nombre de fonctions de hashing utilisées. En effet, les bancs de mémoire des caches associatifs par ensembles utilisent une fonction de hashing commune qui consiste généralement en une simple extraction de bits. Bien que ces fonctions de hashing ne soient pas reconnues comme étant les meilleures (Jain 1992), ils ont l'avantage de diminuer la consommation de mémoire de l'architecture. En effet, puisque'il est possible de récupérer les bits extraits de la clé de recherche à l'aide de la clé de hashing, seuls les bits non-extraits de la clé de recherche doivent être conservés en mémoire.

Certains chercheurs ont quant à eux exploré les différentes possibilités qui permettraient d'optimiser le taux de réussite des insertions. Lim (Lim 2002) propose d'ajouter une table de débordement, qui saurait contenir les éléments causant des collisions pour l'une et l'autre des fonctions de hashing.

La figure 1.7 illustre le cheminement possible d'un élément à insérer dans l'architecture mémoire. Tout d'abord, l'insertion est tentée à l'adresse spécifiée par la première clé de hashing. Dans le cas d'une collision, elle est tentée à nouveau à l'adresse spécifiée par la deuxième clé de hashing. Finalement, s'il y a échec aux deux adresses, l'élément est inséré dans une table de débordement. Lim (Lim 2002) propose d'avoir recours à une unité de CAM pour la réalisation de celle-ci. Lors d'une

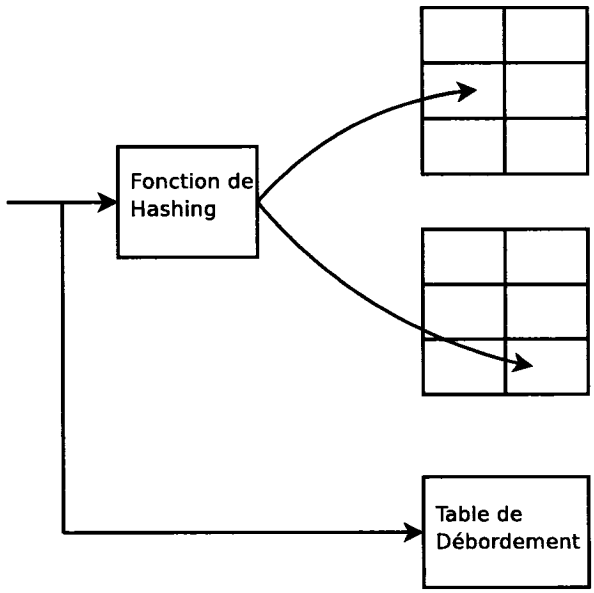

Figure 1.7 Lim (Lim 2002) propose une table de débordement utilisant la technologie CAM pour conserver les données qui n'ont pas pu être insérées.

opération de recherche, trois accès mémoire sont effectués simultanément: deux en mémoire RAM aux adresses spécifiées par les clés de hashing, et une en mémoire CAM. Un circuit logique simple se charge alors de diriger vers la sortie la donnée associée à l'accès réussi s'il y en a un.

Le concept du hashing multiple n'est pas nouveau en soi. Broder (Broder, 1990) a proposé un processus similaire réalisé en logiciel. Cet algorithme consiste en une solution au problème des collisions propre au hashing. L'application réservait un certain nombre de tables et chaque insertion était tentée successivement dans une table après l'autre jusqu'à ce qu'elle soit réussie. Broder propose cependant une technique de re-hashing dans le cas où une insertion cause une collision dans chacune des tables. Cette technique consiste à redéfinir les fonctions de hashing pour que l'ensemble des données puissent être contenues dans les tables. Évidemment, cette technique est très difficilement réalisable en matériel.

Quelques autres pistes architecturales ont été explorées dans le cadre de différents projets. Meribout (Meribout 2003) a conçu une architecture reconfigurable reprenant certains éléments du Cell Processor de IBM. Tarassenko (Tarassenko 1991) a pour sa part étudié la possibilité d'émuler le comportement de la technologie CAM à l'aide de réseaux neuronnaux.

1.3 Positionnement du projet

Les projets de recherche décrits ci-haut proposent des méthodes permettant de réduire certains aspects du coûts de la CAM au détriment d'autres métriques. Comme le lecteur a été en mesure de le constater, la majorité des chercheurs se sont concentrés sur le problème de consommation de la CAM. Les pistes proposées consistent essentiellement en des modifications à la cellule de base de la CAM ou encore en des architectures ayant recours au hashing et à la technologie RAM.

Les solutions avancées représentent des pistes de solution intéressantes. Toutefois, les modifications à la cellule de base de la CAM doivent d'abord être intégrées à un compilateur de CAM avant de pouvoir être intégrées à un design. Ces designs sont ainsi inaccessibles tant et aussi longtemps qu'un concepteur de compilateur de CAM ne les supporte pas.

En ce qui concerne les solutions ayant recours à la technologie RAM et aux techniques de hashing, elles peuvent être facilement intégrées à un design. Leur lacune se situe toutefois dans la rigidité de leur design. Ces alternatives peuvent représenter des solutions très intéressantes, mais seulement pour un ensemble d'applications restreint. En effet, ces designs, tels qu'ils sont présentés, sont inaptes à s'adapter aux besoins très variables des applications. Par exemple, l'architecture de Lim (Seznec 1993) possède deux modules, contenant chacun une clé de hashing et

une unité de mémoire RAM. Certaines applications nécessitent assurément un plus grand nombre de bancs de mémoire. Et pour ce qui est de la table de débordement suggérée par Lim (Lim 2002), la présence d'un bloc de CAM, si petite soit-elle, va à l'encontre d'un des objectifs premiers du présent projet qui est de contourner les frais non-récurrents exigés par les vendeurs de CAM.

Le présent projet retient les avantage des solutions proposant le recours à des bancs de mémoire combinant hashing et technologie RAM, tout en laissant au concepteur le soin de configurer l'architecture selon une granularité très fine en fonction des besoins exacts de son application. En effet, le concepteur pourra décider du nombre de bancs de mémoire et de la taille des cellules de RAM intérieures aux bancs.

Le présent projet offre une méthode facile et rapide d'intégrer à un design une solution alternative à la CAM. Il permet d'obtenir un comportement équivalent à la CAM tout en diminuant la consommation d'énergie et les frais de licences. Ces frais peuvent même être considérés nuls si on assume que les frais associés à la licence d'un compilateur RAM ainsi qu'à un outil de synthèse sont déjà amortis.

CHAPITRE 2

SYNTHÈSE DES TRAVAUX

2.1 Mise en Contexte

Dans le but de concevoir une architecture permettant de réaliser en un seul cycle les opérations d'insertion et de recherche, nous examinons d'abord les différents algorithmes de recherche habituellement associés au domaine du logiciel. Le but de l'exercice est d'étudier la possibilité de réaliser en matériel un de ces algorithmes qui saurait offrir les performances recherchée, ou encore une variante de celui-ci.

Tableau 2.1 Performance des principaux algorithmes de recherche.

Recherche Séquentielle	$\theta(n)$
Arbres Binaires	$\theta(\log n)$
Arbre Balancé	$\theta(\log n)$
Recherche par Racine	$\theta(\log n)$
Correspondance Directe	$\theta(1)$
Hashing	$\theta(1)$
Mémoires CAM	$\theta(1)$

Le tableau 2.1 présente la performance offerte par les principaux algorithmes de recherche, tels que présentés dans la référence (Sedgewick 2002). Il est facile de constater que seuls les algorithmes de hashing et de correspondance directe sont en mesure de rivaliser avec les performances des CAMs. Toutefois, comme il a été mentionné dans la section précédente, la correspondance directe n'est qu'un cas particulier de l'algorithme de hashing qui ne peut être utilisé que pour certaines applications. L'algorithme de hashing apparaît comme étant le candidat le plus prometteur.

L'algorithme de hashing possède des opérations d'insertion et de recherche de complexité $\theta(1)$. Lors d'une opération d'insertion, les données à insérer (les clés de recherche) sont utilisées pour générer l'indice de la table auquel sera effectué l'insertion (la clé de hashing). L'opération de recherche effectue un processus analogue. La comparaison sera faite entre la donnée recherchée et celle présente dans la table à l'indice correspondant à la clé de hashing générée à partir de la donnée recherchée. La fonction de hashing se charge de générer les clés de hashing de manière déterministe. Par exemple, si une fonction de hashing se définit comme effectuant l'extraction des bits 9 à 15 de l'adresse source d'un paquet IPv4, alors tous les paquets provenant de l'adresse "192.168.123.456" seront insérés à l'entrée 123 de la table.

Il est toutefois possible que deux clés de recherche soient associées à une seule et même clé de hashing. Lorsque cette situation survient, on dit qu'il y a collision. Il existe plusieurs méthodes pour gérer les collisions, mais elles nécessitent toutes un nombre d'accès supplémentaires potentiellement grand à la mémoire (Sedgewick 2002).

Le choix du hashing est prometteur, mais la réalisation matérielle se devra de gérer les cas de collision de manière efficace. Lim et Jung (Lim 2002) proposent une approche fort intéressante en ce sens. Celle-ci présente une architecture de hashing parallèle supportant une collision par table et possédant une table de débordement, accueillant les éléments qui n'ont pu être insérés dans les autres tables. Les opérations d'insertion et de recherche sont exécutées en un cycle. Elle tire ainsi profit du parallélisme et des caractéristiques de la technologie RAM pour obtenir des performances équivalentes à celles offertes par la technologie CAM. Il s'agit d'une alternative intéressante qui présente toutefois deux inconvénients majeurs. Tout d'abord, elle ne répond aux besoins que d'un ensemble très restreint d'applications. Sa configurabilité est faible, voire nulle. De plus, elle peut difficile-

ment être considérée comme alternative complète à la CAM, puisqu'elle y a recours pour réaliser la table de débordement. Les coûts non-récurrents de la technologie CAM ne peuvent donc pas être évités.

2.2 Description de l'Architecture

L'architecture présentée dans le cadre du présent projet de recherche reprend quelques concepts à la base de l'architecture présentée par (Lim 2002) sans toutefois figer sa configuration. Elle innove en ce sens qu'elle permet au concepteur de définir sa forme exacte pour répondre de façon optimale aux besoins de l'application. Son plus grand avantage se situe toutefois au niveau de l'absence de toute présence de technologie CAM. En effet, l'architecture de (Lim 2002) a recours à une CAM pour la réalisation de la table de débordement. Même si celle-ci est de petite taille, sa seule présence ajoute les frais de licences au coût total du design.

La figure 2.1 présente l'architecture proposée. Elle permet au concepteur de choisir une configuration éliminant toute possibilité d'échec, ou la réduisant à une valeur négligeable, faisant de l'architecture une alternative de choix pour combler les lacunes de la CAM.

L'architecture consiste en une fonction de hashing associée à chaque table de hashing en plus d'un bloc de logique de contrôle. Lors d'une insertion, la fonction de hashing génère une clé de hashing pour chaque table. L'insertion est alors faite à la table possédant la priorité la plus élevée parmi celles qui ne donnent pas lieu à une collision. Pour ce qui est des opérations de recherche, toutes les tables sont accédées en parallèle. Chacune d'elles compare la donnée accédée à la clé de recherche fournie par l'usager, et envoie le résultat de la comparaison accompagnée par la donnée accédée à la logique de contrôle. Celle-ci filtre les données reçues et

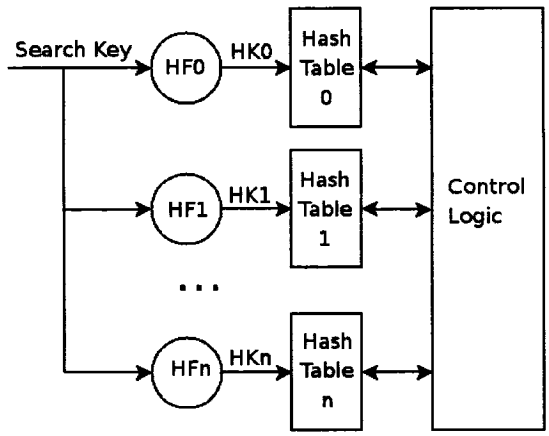

Figure 2.1 Architecture proposée

dirige vers la sortie la donnée accédée par la table à laquelle il y a eu une comparaison réussie. Si aucune table ne signale une comparaison réussie, la logique de sortie signale que l'opération de recherche a échoué.

En plus de laisser tomber la table de débordement pour les raisons décrites ci-haut, l'architecture se distingue de celle décrite par Lim et Jung en (Lim 2002) aussi par l'absence du concept de réservoir (*buckets*). En effet, le concept n'a pas été considéré afin de simplifier leproblème en lui enlevant une de ses variables. À première vue, des gains de performance semblables à ceux réalisés par les réservoirs pouvaient très bien être obtenus en modifiant d'autres paramètres architecturaux sans nécessairement avoir à payer davantage. Une alalyse plus poussée est nécessaire est donc nécessaire pour déterminer s'il d'agit d'un avantage ou d'une limitation. La plus grande différence par rapport à l'architecture décrite en (Lim 2002) réside toutefois au niveau de la configurabilité. L'architecture proposée laisse le soin à l'usager de définir le nombre de couches de l'architecture, tout en lui fournissant les outils pour estimer les performances du système en fonction de l'utilisation de l'application. Le concepteur peut ainsi atteindre les performances que requièrent une application donnée, sans avoir à débourser davantage.

2.3 Évaluation de Performances

Le besoin d'un modèle d'évaluation de performances pour cette architecture provient du fait qu'il est possible qu'une tentative d'insertion résulte en un échec. Cette possibilité est nulle avec la technologie CAM tant et aussi longtemps que l'unité CAM n'est pas remplis à capacité. La présente étude aborde la quantification de performances selon deux métriques: le taux d'échec et la probabilité d'échec des tentative d'insertion.

Le taux d'échecs est défini comme étant le rapport entre le nombre d'insertions résultant en un échec par rapport au nombre de tentatives d'insertion effectuées. Quant à la probabilité d'échec associée à chaque insertion, l'architecture offre des performances qui se dégradent progressivement. Cet aspect fait contraste avec les unités CAM qui acceptent de nouvelles données jusqu'à ce que leur capacité soit atteinte. À ce moment, chaque tentative d'insertion résulte en un échec. Cette différence peut être considérée comme un avantage ou comme un inconvénient, dépendamment de l'application.

Les différentes configurations de l'architecture sont définies à partir de trois paramètres: le nombre de couches, la capacité totale de l'architecture et le taux d'utilisation. À ces paramètres seront respectivement associés les symboles L, C et α dans le cadre des documents relatifs à ce projet de recherche. Le taux d'utilisation, α, est défini comme étant le rapport entre le nombre de données qu'une application vise à emmagasiner dans l'architecture par rapport à la capacité totale de l'architecture.

Le modèle de performances démontre que l'architecture proposée peut offrir des performances de n'importe quel ordre de grandeur, dépendamment de la configuration dans laquelle elle était utilisée. Dans une configuration minimaliste, elle peut agir comme une simple table de hashing avec toutes les limites qu'on lui

connaît. À l'opposé, dans une configuration dégénérée, dans laquelle chaque table est de capacité unitaire, elle peut reproduire les performances exactes d'une unité de CAM.

Il est important de noter que le modèle de performance considère les fonctions de hashing comme étant idéales. Une fonction de hashing idéale disperse de façon parfaitement aléatoire les clés de hashing générées et ce, indépendamment des clés de recherche lues. Dans le cadre d'une application d'aiguillage de paquets par exemple, une fonction de hashing idéale briserait tout lien de corrélation présent dans l'ensemble des adresses réseau actives. Ce genre de corrélation est reconnue comme étant omniprésente et source de plusieurs problèmes de performance dans des designs comme celui-ci. Il s'agit d'une simplification qui nécessite une attention particulière par un concepteur qui considère utiliser l'architecture. La littérature identifie certains algorithmes de CRC comme étant ceux se rapprochant le plus de la définition des fonctions de hashing idéales (Jain 1992). L'implantation faite dans le cadre du présent projet utilise un algorithme de CRC sans toutefois apporter de justifications supplémentaires quant à son choix.

La première constatation se trouve au niveau du taux d'échecs d'une simple table de hashing. Celui-ci semble être constant pour un nombre nombre donné de tentatives d'insertion faites sur des éléments distincts. Lorsque le nombre de tentatives d'insertion est égal à la capacité de l'architecture ($\alpha = 1.0$), le taux d'échecs est de 0.63, une valeur se rapprochant étrangement de $1 - e^{-1}$. Il reste à savoir comment l'ajout de couches et des variation au paramètre α influencent cette valeur de taux d'échecs. C'est une des questions à laquelle tente de répondre le premier article.

2.4 Implantation de l'Arhitecture

Pour ce qui est de l'implantation de l'architecture, nous avons d'abord examiné la possibilité d'avoir recours à un processeur à instructions spécialisées: le XTensa V de Tensilica. Cette voie semblait très prometteuse, d'autant plus que le GRM possède une certaine expertise en la matière ainsi que les licences nécessaires pour utiliser les outils de Tensilica. Le design initial consistait en quelques instructions "TIE" réalisant la fonction de hashing et la logique de contrôle, le tout interagissant avec les unités RAM présentes sur le processeur. Il s'agissait d'un design idéal qui permettait de réaliser l'architecture en logiciel. Toutefois, les contraintes techniques du XTensa ont compliqué les choses. En effet, la cadence du processeur combinée à la quantité maximale de RAM sur la puce ainsi que la taille maximale du bus mémoire limitaient la configurabilité de l'architecture, que l'on voulait sans limite.

L'idée de recourir à des instructions spécialisées n'a pas été écartée immédiatement. Un second design d'implantation a été considéré. Il consistait à sortir du XTensa les tables de hashing et la logique de comparaison. Lors d'une opération de recherche, les instructions spécialisées se chargeaient d'abord d'évaluer les clés de hashing qui étaient transférées à notre ASIC en compagnie de la clé de recherche fournie par l'usager. Les tables de hashing étaient accédées et les données obtenues, comparées à la clé de recherche. Les résultats de la comparaison étaient retournés au processeur qui se chargeait du reste de la logique. Les instructions d'insertion représentaient une plus grande complexité dans ce design puisque le processeur devait initier une deuxième communication avec le ASIC pour lui indiquer l'indice de la table à laquelle devait avoir lieu l'insertion. La présence de deux blocs de logique permettait un certain parallélisme, mais ajoutait toutefois une grande complexité supplémentaire. Encore une fois, ce design a été écarté en raison d'une spécification technique du processeur XTensa. En effet, la largeur maximale de son port IO lim-

ite le nombre de clés de hashing qu'il était possible de transférer du processeur vers le ASIC en un cycle. La configurabilité de l'architecture se voyait encore une fois réduite.

Il a donc été décidé que l'implantation de l'architecture allait se passer de toute forme de processeur et se faire sur un ASIC. Cette décision impliquait l'ajout d'un autre volet à la comparaison avec la technologie CAM, celle du coût en aire d'un bit. En effet, il était pertinent de déterminer si à taille égale, il en coûtait moins d'avoir recours à une unité de l'architecture de hashing parallèle plutôt qu'à une unité de CAM. Si cette hypothèse était validée, il serait alors possible d'affirmer que sans même considérer les coûts non-récurrents de la technologie CAM, l'architecture de hashing parallèle offrait des performances équivalentes à un coût moindre.

2.5 Résumé de l'article

Tout d'abord, les technologies RAM et CAM sont comparées selon différentes métriques. La comparaison révèle que le coût d'aire de silicium en $mm^2/Mbit$ de la CAM est 2 fois supérieur à celui de la RAM, que sa consommation lui est entre 4 fois supérieure. La comparaison faite au niveau de la latence ne montre qu'un léger avantage. La comparaison des métriques ne révèle toutefois qu'une partie de la vérité puisque les frais de licence exigés par les vendeur d'unité issus de la technologie CAM représentes les coûts les plus importants.

L'article définit le comportement interne de l'architecture, permettant au concepteur de la reproduire avec la technologie et les outils qu'il désire. Il aborde ensuite le thème des performances. Pour ce faire, il a été nécessaire de réaliser un simulateur "from scratch" qui reproduisait le comportement de l'architecture. Il est important que l'analyse de ses résultats soit mise en contexte avec l'hypothèse des fonctions

de hashing idéales. En effet, le simulateur implante cette hypothèse à l'aide d'un générateur de nombres aléatoires au niveau des adresses utilisées, éliminant ainsi toutes corrélation possible et permettant à une fonction de hashing très simple de générer des clés de hashing réparties uniformément.

Le simulateur a été utilisé pour illustrer la variation des performances de l'architecture selon un certain nombre de configurations possibles. Les résultats obtenus permettent d'illustrer le comportement exponentiel du décroissement du taux d'échecs lorsque le nombre de couches augmente ainsi que de son accroissement alors que le taux d'utilisation augmente. Ces illustrations fournissent les bases de connaissances aux concepteur désireux de configurer l'architecture en fonction des spécifications d'une application donnée.

L'article fournit de plus une description du comportement de la probabilité d'échec de l'architecture en fonction de l'indice de la tentative d'insertion selon un certain nombre de configurations. L'aspect progressif de la dégradation des performances de l'architecture est un élément clé de leur analyse. En effet, une augmentation du nombre de couches diminue le rythme de l'accroissement de la probabilité d'échec, reportant ainsi l'atteinte d'une valeur non-négligeable à un indice d'insertion plus élevé. Le concepteur peut ainsi juger approprié d'augmenter le nombre de couches tout en diminuant la capacité totale de l'architecture tout en conservant l'atteinte d'une probabilité d'échec critique à un indice d'insertion constant.

Les résultats de simulation illustrent efficacement le comportement de l'architecture selon les différentes configurations dans lesquelles elle peut se trouver. Par contre, elles ne permettent pas d'identifier les configurations menant à des taux d'échecs inférieures à 10^{-4} en raison des temps de simulation trop grands. Or, pour plusieurs applications, un taux d'échecs non-négligeable n'est tout simplement pas acceptable. Pour résoudre ce problème, une première approche étudiée fut de déployer

le simulateur sur une architecture de type "cluster". Cette stratégie aurait permis d'atteindre des ordres de grandeur intéressants. Il était toutefois plus efficace et précis de modéliser de façon analytique le comportement du système pour pouvoir obtenir les métriques de performances de manière analytique. C'est cette dernière approche qui est élaborée dans la seconde publication.

L'article abord ensuite quatre nouveaux thèmes. Il définit d'abord le modèle analytique, pour ensuite présenter une méthodologie pour définir la configuration optimale d'une application donnée. Les métriques de l'implantation faite sur ASIC sont ensuite présentées. Finalement, certaines optimisations mettant l'emphase sur certaines métriques de performance ou de coût au détriment de certaines autres sont décrites.

Le modèle analytique se base sur l'hypothèse de distribution uniforme des clés de hashing, qui assigne une probabilité de $1/m$ à chaque entrée d'une table d'être associée à une clé de recherche donnée où m représente la taille de chaque table. Elle affirme que la probabilité qu'une entrée d'une table soit occupée est égale à $1-P_0$, où P_0 est égal à la probabilité que cette entrée n'ait été associée à aucune clé de recherche sur laquelle une tentative d'insertion a été effectuée antérieurement. Ces données sont suffisantes pour estimer P_0 à l'aide d'une variable aléatoire de Poisson, et permettent ainsi de définir le taux d'occupation local à une table comme étant $1-e^{-\alpha}$. Le taux d'occupation global est ainsi facilement obtenu à l'aide d'une relation de récurrence qui définit le nombre de tentatives d'insertions faite sur une table donnée à partir du nombre de collisions qui ont eu lieu à la table précédente. Cette valeur combinée au nombre de tentatives d'insertion faites dans l'architecture globale permet d'obtenir le taux d'échecs de manière analytique.

En fixant un des paramètres de la configuration et à partir d'un taux d'échecs cible visé par une application, cette méthode permet d'identifier l'autre paramètre

permettant d'atteindre le taux d'échecs visé. Pour les applications ne supportant pas un taux d'échecs non-nul, une règle du pouce utilisée régulièrement est de considérer comme étant négligeable la présence d'un d'un échec durant 10 ans d'utilisation continue de l'application. Cette règle combinée par exemple avec une fréquence d'événements de 100 MHz mène à un taux d'échecs de l'ordre de 3.171×10^{-16}. Cette valeur était inaccessible par le simulateur. Seul le modèle analytique nous permet de l'atteindre.

Théoriquement, il est possible d'identifier une infinité de configurations menant à un taux d'échecs donné. Il est donc à l'avantage d'un concepteur d'identifier la configuration possédant le plus faible coût de production parmi celles menant aux performances visées. L'article propose ainsi une méthode pour effectuer cette tâche. La première étape consiste à définir un modèle de coût à partir des technologies et autres variables propres au projet du concepteur. Ensuite, la méthode amène le concepteur à obtenir la courbe iso de la surface de performance obtenue à l'aide du modèle analytique associant un taux d'échecs à chaque configuration possible. Le minimum de la courbe paramétrique résultant de la projection de la courbe iso sur la surface de coût représente ainsi le coût minimal qu'il est nécessaire de débourser pour atteindre les peformances visées.

L'article présente ensuite un aperçu des métriques obtenues grâce à l'implantation de l'architecture réalisée à l'aide d'outils VLSI. L'utilisation d'un compilateur de cellules RAM a permis d'obtenir des métriques performantes et sur lesquelles il est possible de baser une analyse sérieuse. L'implantation a d'ailleurs eu recours à quelques éléments de la librairie *DesignWare* de Synopsys. La réutilisation de ces blocs réduit l'influence que la qualité du concepteur a sur les métriques obtenues tout en augmentant leur reproductibilité.

La table 2.2 présente le résumé des métriques obtenues suite à la réalisation VLSI

de l'architecture. Les métriques sont assez révélatrices. Elle montrent entre autres qu'à capacité égale, l'aire de la cellule de CAM est 2 fois plus grande, et qu'elle consomme 4 fois davantage d'énergie. Il est normal que ces données soient similaires à celles issues de la comparaison avec la technologie RAM puisque les cellules de RAM occupent plus de 98% de l'aire totale de l'architecture.

Tableau 2.2 Comparaison des métriques de la réalisation VLSI de l'architecture proprosée avec une cellule CAM commerciale. Métrique issues de la technologie 180nm et évaluées avec une fréquence d'opération de 200MHz.

	Architecture de Hashing Parallèle	Portion Occupée par les RAMs	CAM	Facteur CAM / HP
Aire $(mm^2/Mbit)$	8.02	98.10%	19.2	2.4
Puissance $(mW/Mbit)$	309	68.8%	1454	4.7

Il est d'ailleurs intéressant de noter que peu importe le type de contraintes d'optimisation imposée à l'outil de synthèse, l'aire du design final sera occupée en très grande majorité par les cellules de RAM. Faire passer la fréquence d'horloge maximale d'une valeur inférieure à 40 MHz à une valeur supérieure à 200 MHz augmente l'aire de la logique combinatoire par un facteur de 40%. Par contre, la portion occupée par cette logique est tellement faible par rapport à l'aire totale que ces modifications n'ont pratiquement aucune influence sur l'aire totale et sur la portion occupée par les cellules RAM, qui elle occupent plus de 98% du design. Il est intéressant de noter que plus de 65% de la période d'horloge optimale est consommée par les accès parallèles aux cellules de RAM. Il serait ainsi possible d'obtenir une fréquence d'horloge beaucoup plus grande en ayant recours à des cellules de RAM plus rapides. La table 2.3 illustre la situation.

Finalement, l'article présente trois optimisations applicables à l'implantation originale. Tout concepteur se doit de les considérer et de les évaluer en fonction des besoins de l'application.

L'implantation initiale utilise des cellules RAM ne possédant qu'un seul port de lecture et d'écriture. Elle est considérée comme étant "multicycle" puisqu'elle réalise les opérations de lecture en un cycle et celles d'écriture, en deux. Cet aspect est reconnu comme étant source de maints problèmes de synchronisation. La première optimisation résout cet inconvénient en ayant recours à des cellules RAM possédant deux ports. Elle cache le second cycle des opérations d'écriture en utilisant le deuxième port des cellules de RAM pour réaliser l'insertion en mémoire, seule opération effectuée dans le second cycle d'une opération d'insertion. Il apparaît ainsi à l'usager que l'architecture est prête à réaliser une nouvelle opération dès le cycle suivant. Cette optimisation permet d'augmenter le débit des opérations et de diminuer leur latence moyenne. Il en résulte aussi une augmentation de l'aire de sillicium requise et de la consommation d'énergie. En effet, les cellules RAM à deux ports consomment davantage que les cellules RAM à port unique et occupent près du double de leur aire.

La seconde optimisation consiste à pipeliner le design pour ainsi atteindre un débit d'opérations plus grand. La période d'horloge serait définie par l'opération atomique la plus lente, soit l'accès aux cellules de RAM, qui dans notre cas demande 2,8ns. Le pipeline nécessiterait un étage de pré-traitement et un de post-traitement en plus de l'étage d'accès aux mémoires. Cette optimisation, combinée à à la

Tableau 2.3 La diminution de la période d'horloge d'influence pas la portion de l'aire occupée par les cellules de RAM.

Période d'horloge (ns)	Aire Totale (sq μm)	Aire de la logique Combinatoire (sq μm)	Portion occupée par les RAMs
26.98	1,699,135	22,955	98.65%
9.48	1,699,518	23,338	98.63%
4.35	1,704,266	28,086	98.35%
4.27	1,707,043	30,863	98.19%
4.24	1,708,629	32,449	98.10%

précédente, permettrait d'atteindre un débit supérieur à 350×10^6 operations par seconde.

La troisième et dernière optimisation permet de minimiser la consommation de puissance de l'architecture en effectuant les recherches dans lec couches non plus de facon simultanee, mais bien de facon sequetielle. Les couches sont rassemblees en groupes et ces groupes sont accedés les uns après les autres. Pour des valeurs de $\alpha < 1$, les tables supérieures contiennent un nombre d'éléments plus grand que les tables inférieures. Il est ainsi plus probable de retrouver un élément recherché dans une des premières tables plutôt que dans un des dernières. Puisque les tables qui ne sont pas accédées ne consomment pas, il en résulte une économie de consommation de puissance. Cette économie est évidemment réalisée aux dépens d'une augmentation de la latence moyenne.

CHAPITRE 3

RÉALISATION VLSI DE L'ARCHITECTURE

3.1 Méthodologie

Pour réaliser l'implantation VLSI de l'architecture, l'équipe a choisi de développer en parallèle ses modèles VHDL et SystemC et d'avoir recours à un simulateur mixte. Cette façon de faire permet de définir les cas de tests en SystemC, de les exécuter sur les deux modèles, de valider le modèle VHDL en comparant son comportement avec celui du modèle SystemC. Les deux modèles sont considérés valides si leur comportement est identique pour l'ensemble des tests. Une fois cette étape franchie, le modèle synthétisé se doit à son tour passer la batterie de tests pour s'assurer qu'aucune contrainte de délai n'est violée.

3.2 Description Comportementale

L'architecture possède 5 signaux d'entrée et deux en sortie. L'usager qui désire effectuer une opération d'insertion doit tout d'abord présenter à l'entrée de la cellule la clé de recherche (SK) accompagnée de la donnée qui lui est associée (PAYLOAD) sans oublier de lever le signal d'écriture (W). Ces signaux doivent être valides au moment du front montant de l'horloge (CLK). Lorsque l'opération d'insertion est complétée, le signal MF indique si l'architecture a réussi ou non à insérer la donnée en mémoire. La valeur du signal PAYLOAD_O est indéterminé suite à une insertion.

Pour ce qui est d'une opération de recherche, l'usager doit tout d'abord présenter à l'entrée de la cellule la clé recherchée (SK) en prenant soin d'abaisser le signal d'écriture (W). Ces signaux doivent être encore une fois valides au moment du front montant de l'horloge (CLK). La valeur présente sur PAYLOAD n'est pas lue lors d'une opération de recherche. L'architecture signale une recherche réussie si la donnée se trouve dans un des étages et si elle y est présente depuis un nombre de cycles inférieur à la valeur présente sur le signal TS_TTL. En effet, l'usager peut spécifier une durée maximale après laquelle toute donnée en mémoire est périmée. Lorsque cette étape est complétée, le signal SK indique à l'usager si l'opération de recherche est réussie. Si c'est le cas, le signal PAYLOAD_O présente à l'usager la donnée associée à la clé recherchée. Dans le cas contraire, sa valeur est indéfinie.

3.3 Fonctionnement Interne

La réalisation VLSI a été faite à partir du design multi-cycles de l'architecture. Les opérations de lectures s'y effectuent en un cycle tandis que celles d'écriture en nécessitent deux. Durant le premier cycle d'une opération d'insertion, le bloc HF génère la clé de hashing à partir de la clé de recherche présentée par l'usager. Cette clé de hashing est divisée parmi toutes les couches et fournit à celles-ci l'information nécessaire pour identifier le seul indice de la table dans lequel la donnée à sauvegarder peut être déposée. L'identité de la couche à laquelle aura lieu l'insertion nécessite toutefois l'accès à la mémoire. En effet, l'insertion aura lieu à la couche la plus prioritaire parmi celles où l'insertion ne cause pas de collision. Les tables sont donc accédées à l'indice spécifié par la clé de hashing qui leur est associée. Chaque couche informe le bloc OUT_LOGIC de la disponibilité de l'entrée accédée. Une entrée en RAM est considérée disponible si le bit PRESENT est à 0 ou encore si la durée écoulée depuis la dernière insertion de cette donnée excède

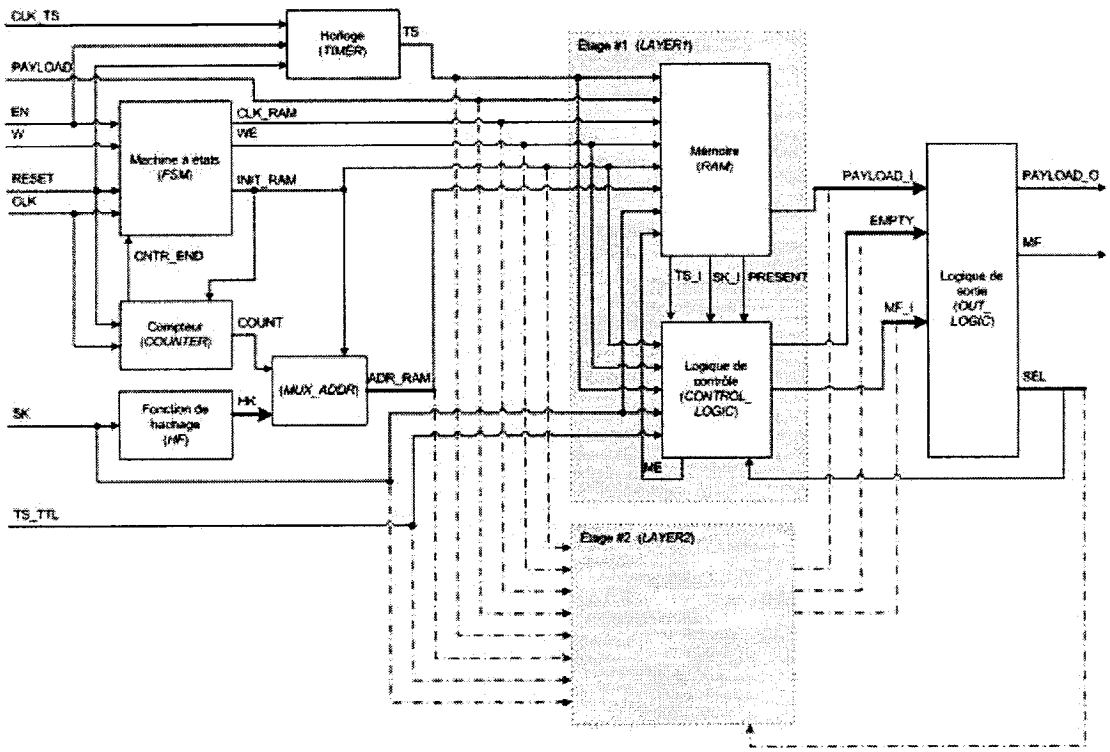

Figure 3.1 Schéma bloc de l'architecture

la valeur limite spécifiée par l'usager à l'aide du signal TS_TTL. Ce bloc rassemble les informations reçues et identifie la couche à laquelle aura lieu l'insertion à l'aide du signal SEL selon l'algorithme décrit précédemment. Durant le second cycle de l'opération d'insertion, la couche choisie par le bloc OUT_LOGIC procède à l'écriture en RAM de la données à insérer. La rangée de RAM écrite verra son bit présent mis à 1.

L'opération de lecture est pratiquement identique à la première partie de l'opération d'écriture. Dans cette opération, chaque couche reçoit la clé de hashing fournie par le bloc HF. Les tables sont alors accédées à l'adresse associée à indice qu'elles ont reçu et les données lues sont comparées à la clé recherchée fournie par l'usager. Le résultat des comparaisons accompagnés de la portion utile (payload) des données

accédées en mémoire sont transmis au bloc OUT_LOGIC. Si une des couches signale une comparaison réussie, ce bloc se charge de lever le signal MF et de rediriger en sortie la portion utile provenant de cette couche. Dans le cas où aucune couche ne signale une comparaison réussie, il se content de garder le signal MF bas.

Bloc RAM

Chaque bloc RAM consiste en une cellule RAM à un seul port configurée selon les spécifications demandées par l'usager. Chaque entrée est divisée en trois champs, tel qu'illustré à la figure 3.2. Tout d'abord, le bit PRESENT indique si l'entrée est occupée ou non. Ensuite, le champs SEARCHKEY contient la clé de recherche sur laquelle va être effectuée une comparaison lors de la recherche. Finalement, le champs PAYLOAD contient la donnée utile associée à la clé de recherche.

Figure 3.2 Disposition des champs en RAM

La taille des champs dépense de l'application. Par exemple, dans le cadre d'une application d'aiguillage de trames Ethernet à 8 ports, les rangées des unités de RAM possèderaient 52 bits chacunes: 1 pour le champs présent, 48 bits pour contenir l'adresse MAC dans le champs SEARCHKEY et 3 bits pour identifier le port permettant de rejoindre cette adresse.

Bloc FSM

Le bloc FSM gère l'état du système. Il s'agit en fait d'une simple machine à 4 états dont les transitions sont définies à partir de l'utilisation faite par l'usager. Elle

définit tout d'abord un état d'initialisation, INIT, dans lequel le système pénètre lorsque le signal RST est bas et dans lequel le contenu des RAM se voit effacé. La machine quitte cet état lorsque le bloc COUNTER signale la fin de la routine d'initialisation en levant le signal CNTR_END.

Quand à l'état IDLE, il est actif quand le signal EN est haut. Les deux autres états, READ et WRITE, coordonnent les opérations de recherche et d'insertion. Lorsqu'une opération est amorcée par l'usager, le bloc FSM est dans l'état READ. Au cycle suivant, la machine se déplace à l'état WRITE si l'opération en cours est une insertion. Dans le cas contraire, l'état reste à READ et la cellule est prête à exécuter une nouvelle opération.

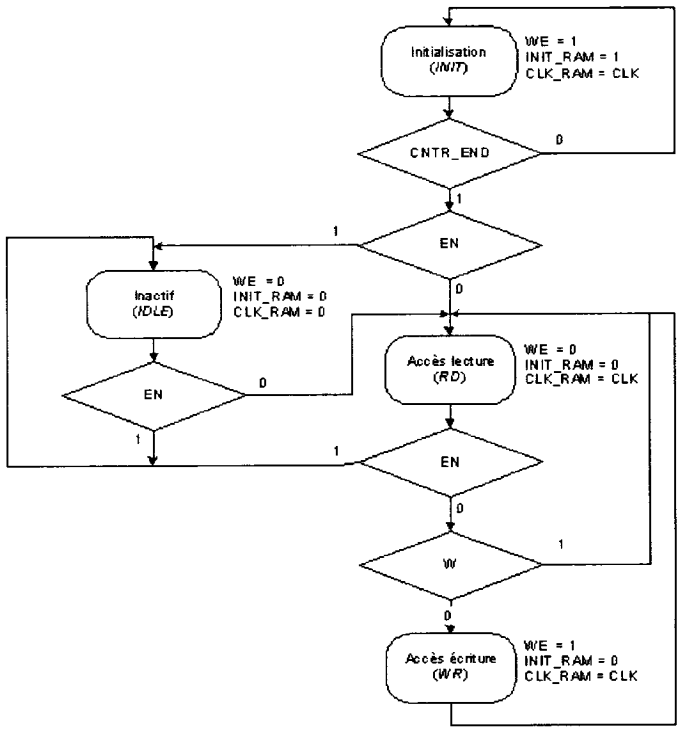

Figure 3.3 Machine à États

Bloc COUNTER

Le bloc COUNTER génère les adresses mémoire utilisées pendant l'initialisation. Il s'agit d'un simple compteur qui parcourt l'ensemble des adresses des cellules de RAM et qui lève le signal CNTR_END lorsqu'il y a débordement.

Bloc HF

Le bloc HF réalise l'algorithme de fonction de hashing. Il se charge de générer de façon déterministe les bits des clés de hashing de l'ensemble des couches à partir de la clé de recherche présentée par l'usager. La clé de recherche est ainsi associée à un indice dan chaque table de hashing.

Le choix de l'algorithme représente quant à lui un choix de design majeur. Il joue un rôle crucial au niveau des performances de l'architecture. Toutefois, aucun algorithme n'est reconnu comme étant le meilleur à ce titre. Son choix doit donc faire l'objet d'une attention particulière et être présent lors de la présentation de quelconque forme de présentation de performances associées à l'architecture.

Le présent projet suit les recommandations décrites en (Jain 1992) et utilise l'algorithme de CRC pour la génération des clés de hashing. Cet algorithme possède toutefois plusieurs variantes et chacune d'elle représente un choix qui peut avoir de lourdes répercussions sur les performances de l'architecture. La littérature nous présente à ce sujet plusieurs analyses identifiant les meilleurs polynômes générateurs permettant d'identifier la présence d'erreurs de transmission et de les corriger (Koopman 2004). La capacité de reconnaître une erreur de transmission et celle de disperser uniformément des clés de hashing à partir de clés de recherche corrélées entre elles sont toutefois des qualités distinctes propres aux polynômes générateurs. Pour

cette dernière caractéristique, la littérature ne nous est malheureusement que très peu utile: il semble que nous soyons les premiers à explorer ce territoire.

Une étude entière aurait été nécessaire pour identifier de façon rigoureuse un polynôme générateur permettant de réaliser une fonction de hashing s'approchant de la fonction de hashing idéale. Dans le cadre du projet d'implantation VLSI du design, les bits utilisés pour générer les clés de hashing sont issus de blocs CRC configurés selon les polynômes présentés en (Koopman 2004). L'équipe s'assure ainsi d'un minimum de performances tout en permettant d'estimer de façon réaliste les métriques associées à la réalisation matérielle du design: aire, dissipation et latence.

Bloc CONTROL_LOGIC

Le bloc CONTROL_LOGIC est présent dans chaque couche de l'architecture et se charge de gérer les accès à la cellule RAM. Tout d'abord, il contrôle le signal ME (memory enable) de la cellule RAM qui lui est associée. Lors des opérations de recherche, dans lesquelles toutes les cellules RAM sont accédées simultanément, il le conserve à '1'. Lors d'une opération d'insertion, il ne ne permettra l'écriture à la cellule RAM qu'à la condition qu'il redoive l'autorisation du bloc OUT_LOGIC via le signal SEL.

Le bloc se charge de la logique propre à chaque couche. Lors d'une recherche, le bloc s'occupe d'effectuer la comparaison entre la clé de recherche lue en RAM et celle fournie par l'usager pour permettre au bloc OUT_LOGIC de déterminer si la donnée recherchée est présente dans la couche. Lors d'une insertion, il redirige en sortie le bit PRESENT lu en RAM pour que le bloc OUT_LOGIC puisse déterminer si l'insertion à la couche causerait une collision.

Bloc OUT_LOGIC

Les fonction du bloc OUT_LOGIC consistent à réaliser la logique commune aux couches en plus de générer les signaux de sortie.

Lors d'une opération de recherche, elle rassemble les signaux MF_I, indiquant pour chaque couche si celle-ci contient la donnée recherchée. Elle se charge de lever le signal de sortie "match flag"(MF) si au moins une des couches indique avoir trouvé la donnée recherchée. Le cas échéant, le bloc redirige en sortie la portion PAYLOAD de la couche en question.

Lors d'une opération d'insertion, le bloc se charge de rassembler les signaux indiquant la disponibilité de la rangée associée à la donnée à insérer provenant de chaque couche. Il est ainsi en mesure de déterminer si l'insertion est possible. Si c'est le cas, le bloc autorise l'écriture à la couche la plus prioritaire parmi celles affichant une entrée RAM disponible en levant son signal SEL.

Bloc MUX_ADDR

Le bloc MUX_ADDR se charge d'identifier la source des données utilisées pour les adresses RAM. Quand le système est en cours d'initialisation, le bloc redirige la sortie du bloc COUNTER vers chacune des cellules RAM. De cette façon, les entrées RAM sont effacées séquentiellement. Durant l'utilisation normale du système, le bloc laisse passer les clés de hashing vers les adresses des cellules RAM pour que les opérations de recherche et d'insertion puissent se réaliser correctement.

3.4 Choix d'implantation

Une fois la définition des tâches de chaque bloc complétée, plusieurs décisions devaient être prises relativement à leur réalisation au niveau RTL. La première d'entre elles consistait à identifier la technologie utilisée.

La présentation d'une architecture alternative à la CAM n'a que très peu de valeur si elle n'est pas accompagnée de fiches comparatives. En effet, tout concepteur serait très hésitant à y avoir recours sans savoir comment l'architecture se compare à la CAM au niveau des métriques essentielles incluant son coût en aire et sa dissipation de puissance. Ces métriques étant intimement liées avec la technologie utilisée, il est donc préférable que les fiches comparatives soient toutes basées sur la même technologie. Bien que certaines méthodes permettent de comparer des métriques issues de technologies différentes, elles possèdent un grand degré d'imprécision et enlèvent du sérieux à la comparaison. La technologie 0.18 μm fut ainsi choisie en raison de la possibilité d'y avoir recours pour réaliser l'ensemble des fiches comparatives. En effet, notre partenaire industriel nous donnait accès à des fiches techniques de cellules CAM en 0.18 μm. De plus, cette technologie nous était très facile d'accès via les outils disponibles au GRM.

La reproductibilité représentait un autre critère d'importance dans la réalisation RTL de l'architecture. En effet, l'équipe s'était donné comme mission de réduire au minimum l'influence des aptitudes de chacun sur les métriques obtenues, quitte à ce que celles-ci soient légèrement inférieures en bout de ligne. Pour ce faire, les éléments de la librairie DesignWare de Synopsys allaient être utilisés lorsque possible.

Le bloc qui se prête le mieux à l'utilisation d'éléments de la librarie DesignWare est sans aucun doute le bloc HF. En effet, la qualité de réalisation RTL d'algorithmes de

CRC est fortement dépendante vis-à-vis sont concepteur. Il s'agit d'un algorithme complexe qui peut être réalisé de plusieurs façons, chacune desquelles pouvant aboutir à une combinaison de métriques différente. Pour annuler les effets de ces variables, le module de génération combinatoire de CRC de la librairie DesignWare fut utilisé. Il supporte un grand nombre de variantes de l'algorithme de CRC via ses paramètres de configurations qui permettent entre autres de définir la largeur du mot d'entrée, le polynôme générateur ainsi que la valeur initiale du registre de décalage interne. Il était ainsi possible de reproduire intégralement l'algorithme de CRC décrit en (Koopman 2004) sans avoir à apporter quelconque modification manuelle au bloc.

Parmi les autres modules ayant recours aux éléments de la librairie DesignWare, on retrouve les blocs TIMER, COUNTER et MUX_ADR. Ces blocs utilisent des circuits logiques de base bien connus. Les deux premiers ont recours à un compteur tandis que l'autre utilise un multiplexeur. Pour le bloc OUTPUT_LOGIC, cependant, une logique "custom" a été préférée aux blocs de DesignWare. Lors d'un accès en recherche, ce bloc se doit de laisser passer en sortie la portion PAYLOAD de la données en RAM provenant de l'étage où a lieu une comparaison réussie. Il aurait été possible de réaliser ce bloc à l'aide d'un encodeur de priorité combiné à un multiplexeur. La figure 3.4 illustre le concept. Les signaux "match flag" provenant de toutes les couches entrent dans l'encodeur de priorité. Celui-ci dépose en sortie l'identité de l'étage le plus prioritaire parmi ceux signalant une comparaison réussie. Cette valeur est déposée comme signal de sélection à l'entrée du multiplexeur qui aiguille le signal PAYLOAD de la couche correspondante.

Cette réalisation aurait certes été fonctionnelle. Par contre, comme il a été mentionné précédemment, il est impossible que plus d'un étage soit lieu d'une comparaison réussie. L'encodeur de priorité traite ainsi un grand nombre de cas qui ne surviendront jamais pendant une utilisation normale de l'architecture. La solution

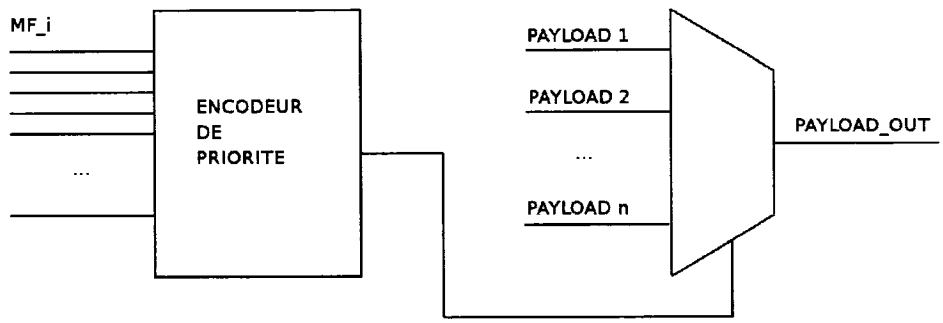

Figure 3.4 Circuit de sélection du signal PAYLOAD réalisé à l'aide d'un encodeur de priorité

présentée à la figure 3.4 est donc beaucoup trop lourde. Une solution considérant la prémisse décrite ci-haut représenterait une solution beaucoup mieux ajustée au problème. La solution retenue a recours à un ensemble de portes de transmission et permet d'économiser un grand nombre de portes logiques.

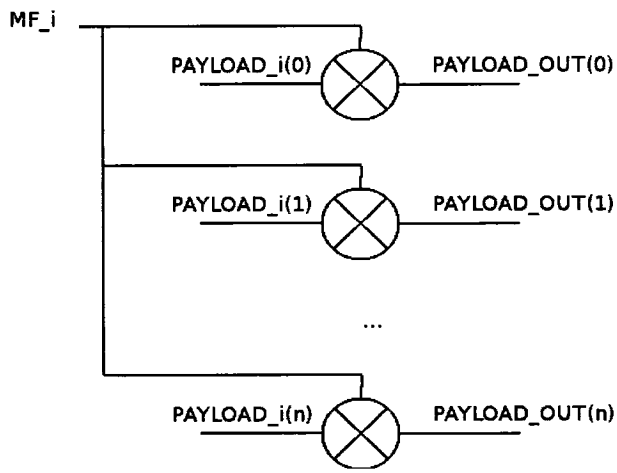

Figure 3.5 Circuit de sélection du signal PAYLOAD réalisé à l'aide de portes de transmission

La figure 3.5 illustre le circuit de sélection du signal PAYLOAD réalisé à l'aide de portes de transmission. Ce circuit est répété pour chaque couche présente dans l'architecture. Et puisque le signal $PAYLOAD_OUT(i)$ ne sera actif que pour une

seule valeur de i, il est permis de joindre les signaux PAYLOAD_OUT correspondants et les présenter directement en sortie. L'économie est donc substantielle, puisqu'une porte de transmission ne consomme que 2 transistors (Savaria 1988).

3.5 Simulations et Vérifications

La stratégie de vérification consiste en premier lieu à valider le modèle fonctionnel décrit dans l'environnement SystemC. Une fois validé, ce modèle nous permet de vérifier le bon fonctionnement de la réalisation VLSI de l'architecture. Le processus est illustré à la figure 3.6.

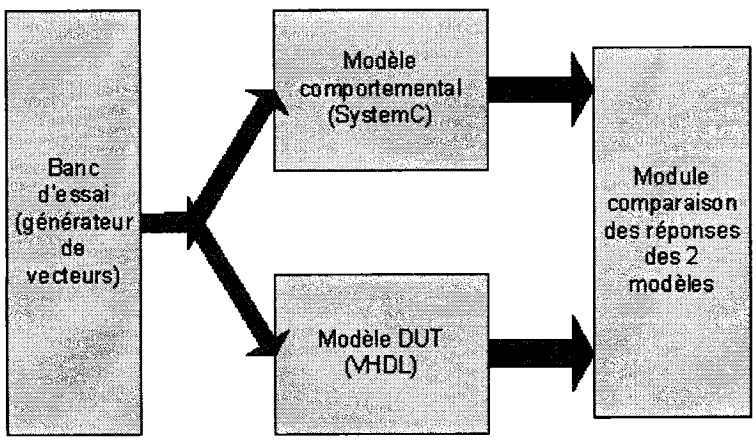

Figure 3.6 Stratégie de simulations mixtes

L'outil "ncsim" de Cadence nous permet d'effectuer la simulation mixte nécessaire à la méthodologie de vérification. Ce processus consiste à réutiliser le même banc de tests pour tous les modèles du DUT. Il est ainsi possible de vérifier le bon fonctionnement des modèles RTL pré-synthèse et post-synthèse en comparant leur comportement à celui du modèle comportemental décrit en SystemC. Le fait de réutiliser le même banc de tests permet d'éliminer toute possibilité d'erreur dans la traduction du banc de tests d'un langage à un autre. Il est ainsi possible d'attribuer

la cause de différences observées au niveau du comportement des modèles de DUT à des erreurs de design du modèle RTL. La simulation mixte permet ainsi de sauver une grande quantité de temps à l'étape de vérification.

Avant même d'amorcer la réalisation VLSI de l'architecture, l'équipe a défini un ensemble de tests que devait passer le modèle RTL de l'architecture et de chacun des blocs pour que celle-ci puisse être jugée fonctionnelle. Ces tests couvraient un certain nombre de cas d'utilisation normale en plus de certains cas limites. Un test était considéré comme étant réussi si le comportement du modèle RTL correspondait à celui du modèle SystemC.

3.6 Résultats

Les unités de RAM utilisées dans le présent design sont de dimension 1024x76 et occupent 575916 sq-microns en technologie 180nm. Sans appliquer aucune contrainte, la synthèse de Design Compiler a nécessité 2,339e6 sq-microns pour réaliser une configuration de 4 étages de notre design. C'est donc dire que les RAM occupent 98,5% de l'aire de la cellule. Il est conséquemment à notre avantage de demander à Design Compiler de diriger l'optimisation de la synthèse de notre logique sur les délais puisque l'accroissement de l'aire serait alors négligeable par rapport à l'aire globale. L'outil a ainsi réussi atteindre un délai maximal entre deux registres séquentiels de 2,73ns. L'aire totale est alors évaluée à 2,360e6, ce qui ne représente qu'une augmentation 0,5%.

Ce délai cache toutefois une grande partie du problème. En effet, la principale contrainte se situe au niveau du temps de pré-positionnement de la clé de recherche. Le bloc HF, chargé de réaliser l'algorithme CRC en un seul cycle et de générer les clés de hashing, comporte un délai important. Un bloc CRC synchrone aurait réalisé

l'algorithme en réutilisant les mêmes portes à chaque cycle. Le bloc CRC combinatoire, quant à lui, réalise la même opération en un seul cycle en dupliquant ses portes et en les disposant les une après les autres. Il en résulte un chemin critique important. La figure 3.7 présente la simulation pré-synthèse du bloc CRC combinatoire de la librairie DesignWare tandis que la figure 3.8 présente la simulation post synthèse. Cette figure démontre un délai de 11ns nécessaire à la génération d'un CRC de 10 bits! Les optimisations de synthèse ont heureusement permis de réduire ce délai à une valeur de 4ns.

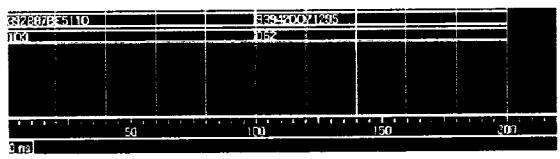

Figure 3.7 Simulation pré-synthèse du bloc HF

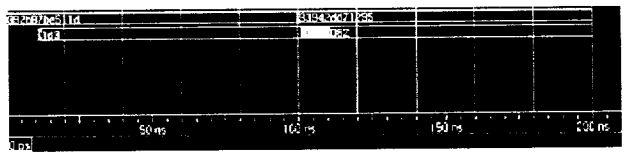

Figure 3.8 Simulation post-synthèse du bloc HF

L'évaluation de la fréquence maximale peut se faire de deux façons. Tout d'abord, dans le cas d'une horloge continue, la fréquence maximale ne pourra dépasser l'inverse de la somme du temps de pré-positionnement et du délai maximal entre deux registres synchrones. Ainsi, peu importe si l'architecture est dans un état d'initialisation, de lecture ou d'écriture, aucune contrainte se sera violée. Toutefois, si l'horloge est contrôlée par un processeur maître, celui-ci pourrait ajuster la période d'horloge en fonction du traitement effectué. Les cycles d'écriture pourraient ainsi être exécutés plus rapidement puisqu'ils ne demandent pas la génération de clés de hashing. Pour les fins de comparaisons de métriques, la fréquence

maximale se doit d'être basée sur une horloge continue. Il en résulte une comparaison plus honnête puisque les métriques de la technologie CAM ne possèdent pas de contraintes de gestion de la période de l'horloge. La fréquence maximale de l'architecture peut donc être identifiée comme étant 150 MHz. Il est toutefois important d'accompagner cette valeur de certaines remarques. Tout d'abord, il est nécessaire de rappeler le caractère multi-cycle de l'architecture, l'écriture nécessitant 2 cycles. De plus, le débit des données peut être amélioré de plusieurs façons. La première consiste à utiliser un contrôleur d'horloge, ajustant la période d'horloge tel que décrit précédemment. Autre autre est décrite dans la section précévente et consiste à recourir à des cellules RAM à deux ports permettant de réaliser les écritures en un seul cycle. Finalement, il serait possible de pipeliner l'architecture en limitant le niveau de logique présente entre chaque paire de registres pour atteindre un débit encore plus élevé.

En ce qui concerne la dissipation de chaleur, il serait imprudent de se baser sur les valeurs fournies par l'outil de synthèse parce que cette métrique est fortement liée avec l'utilisation qui est fait de la cellule. Une simulation post-synthèse, voire post-placement et routage permettrait d'obtenir des évaluations plus crédibles.

CHAPITRE 4

DISCUSSION GÉNÉRALE

Le présent mémoire a exposé au lecteur les lacunes majeures de la technologie CAM: aire de silicium et consommation d'énergie. Il a présenté un résumé des efforts de recherche réalisés dans le but de pallier à ces lacunes. Plusieurs d'entre eux offrent des gains appréciables sans pour autant combler les besoins des designs actuels. En effet, les modifications apportées à la cellule de base nécessitent d'abord d'être intégrées à un compilateur de CAM avant qu'elles puissent être utilisées dans un design. Et les architectures proposées émulant le comportement d'une CAM ne répondent aux besoins que de certaines applications.

Le présent mémoire propose une solution offrant des avantages marqués sur les solutions alternatives disponibles. La solution proposée présente une architecture qu'il est possible de facilement et rapidement intégrer à un design. Le mémoire fournit de plus les outils nécessaires au concepteur pour lui permettre de configurer les paramètres de l'architecture afin que celle-ci offre une combinaison optimale de coût et de performances en fonction d'une application donnée.

Le modèle analytique permet quant à lui d'identifier l'ensemble des configurations menant à une performance donnée. De plus, si le concepteur possède un modèle de coûts, il est alors possible d'identifier la configuration à coût minimal menant à la performance visée.

Les seuls outils nécessaires à l'intégration de cette architecture dans un design sont un compilateur de RAM ainsi qu'un outil de synthèse. Puisque la grande majorité des designs d'une certaine envergure nécessitent déjà le recours à ces outils, il est

permis d'affirmer que l'insertion de l'architecture dans un design n'implique pas de coûts additionnels.

Deux autres modifications à l'architecture ont été présentées dans le cadre du second article. La première consiste à avoir recours à des cellules RAM à deux ports. Ceci permettrait de dédier le second port aux écritures pour qu'elles puissent être réalisées à l'intérieur du même cycle que l'opération de lecture de l'accès mémoire suivant. Cet important gain réalisé au niveau du temps d'accès moyen est obtenu en échange d'un accroissement de l'aire occupée par les cellules de RAM. En effet, les cellules RAM à deux ports occupent un peu moins que le double de l'aire des cellules RAM à un port équivalentes.

La seconde optimisation consiste à regrouper les couches et accéder séquentiellement aux groupes lors des opérations de recherche jusqu'à ce que l'élément recherché soit trouvé ou que le dernier groupe soit accédé. Cette optimisation se base sur le fait que la probabilité que la donnée recherchée soit présente à l'intérieur d'un groupe donné baisse au fur et à mesure que l'indice du groupe augmente. Autrement dit, si elle est présente, la donnée a une plus grande probabilité de l'être à l'intérieur de l'un des premiers groupes. Ainsi, dans le cadre de cette optimisation, une majorité des accès n'auront pas à descendre jusqu'aux groupes inférieurs, économisant ainsi une grande proportion de l'énergie consommée lors des accès aux blocs de mémoire RAM.

CONCLUSION

Le concepteur n'est pas limité au design de l'architecture tel que défini dans ce document. Il est invité à y apporter quelconque améliorations dont pourrait bénéficier son application. Tout d'abord, il serait possible de rapatrier le concept de réservoir, paramètre de configuration qui a été écarté dans les premiers moments du projet. Celui-ci consiste à permettre à chaque rangée de RAM de contenir plus d'un élément. Sa réalisation pourrait être faite en élargissant les blocs RAM ou encore en juxtaposant plusieurs blocs de RAM l'un à côté de l'autre à l'intérieur d'une même couche. L'impact sur les performances pourrait être facilement défini en apportant une légère modification au modèle analytique. L'impact sur le coût, quant à lui, pourrait être plus difficile à cerner. En effet, plusieurs facteurs entrent en jeu. Par exemple, la variation de l'aire des cellules de RAM utilisées dans le design en fonction de l'accroissement de la largeur de la cellule sera un facteur déterminant.

Le lecteur intéressé est aussi invité à mettre l'épaule à la roue et approfondir les recherches du présent projet. Une façon de faire serait de raffiner les différentes métriques présentées dans cet ouvrage. L'estimation de la latence ne tient compte que d'une approximation des résistances parasites. Or, ces valeurs sont intimement liées avec les distances entre les portes, données inconnues avant la synthèse. L'évaluation de l'aire, quant à elle, n'est le résultat que d'un facteur appliqué au nombre de portes. Finalement, l'évaluation de la dissipation de puissance est le fruit de la somme des données fournies par le vendeur des cellules de RAM et du nombre de portes auquel un facteur propre à la technologie utilisée a été appliqué. Il serait possible d'obtenir des métriques beaucoup plus précises et fiables si le placement et routage d'une configuration de l'architecture était réalisée. La latence serait alors basée sur les distances réelles entre les portes et l'estimation de l'aire tiendrait compte des vraies interconnexions reliant les portes. Finalement,

un simulateur pourrait alors se charger de fournir une estimation post-placement-routage, données qui donnerait un tout autre poids à l'architecture proposée dans ce projet de recherche.

Le lecteur soucieux d'approfondir le présent projet pourrait aussi s'inspirer des travaux de (Liu 2001) et proposer une modification à l'architecture lui permettant de définir des fonctions de hashing configurables à partir de signaux d'entrée. Il serait alors possible qu'un acteur extérieur prenne connaissance des données que l'application entend entreposer dans l'unité mémoire. Cet acteur pourrait alors identifier des fonctions de hashing qui sauraient réduire le nombre d'échecs, voire les éliminer complètement.

Une toute autre voie qu'il serait intéressant d'approfondir est celle concernant l'évaluation de performance des fonctions de hashing basées sur l'algorithme de CRC. Celui-ci est généralement reconnu comme étant le meilleur pour s'approcher la fonction de hashing idéale. La littérature ne fournit toutefois pas les outils permettant de choisir les polynômes générateurs les plus performants pour une application donnée. En effet, elle est très généreuse au sujet de la capacité des polynômes à détecter la corruption de données. Toutefois, cette caractéristique est orthogonale avec celle permettant de briser toute corrélation entre un trafic donné et l'ensemble des clés de hashing générées. Le concepteur ne possède donc pas les outils pour concevoir une architecture optimale. Il doit se prêter à un jeu d'essais et erreurs et ainsi tenter de trouver les combinaisons permettant à l'architecture d'approcher les performances théoriques.

RÉFÉRENCES

BRODER, A. and KARLIN, A. 1990. Multilevel adaptive hashing. In Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. ACM, San Francisco, California, United States, 43-53.

DELGADO-FRIAS, J.G. NYATHI, J. TATAPUDI, S.B. 2005. Decoupled dynamic ternary content addressable memories. Circuits and Systems, IEEE Transactions on 52, 10 (October), 2139-2147.

EFTHYMIOU, A. and GARSIDE,J. D. 2002. An adaptive serial-parallel cam architecture for low-power cache blocks. In Proceedings of the 2002 international symposiumon Low power electronics and design. 136-141.

HENNESSY, J. L. PATTERSON, D. A. Computer Architecture: A Quantitative Approach, 3rd edition. Morgan Kaufmann. 2003.

JAIN, R. 1992. A comparison of hashing schemes for address lookup in computer networks. Communications, IEEE Transactions on. 1570-1573.

KARTALOPOULOS, S. Associative RAM-based cam applicable to packet-based boradband systems. In Global Telecommunications Conference, 1998. Sydney, NSW. Pages 2888-2891.

KOOPMAN, P. and CHAKRAVARTY, T. Cyclic redundancy code (crc) polynomial selection for embedded networks. In Dependable Systems and Networks, 2004 International Conference on, pages 145-154.

LIM, H. and JUNG, Y. 2004. A parallel multiple hashing architecture for ip address lookup. High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on, 91-95.

LIM, H. SEO, J-H. JUNG, Y-J. 2003. High speed ip address lookup architecture using hashing. IEEE Communications Letters. 502-504.

LIN, C-S. CHANG, J-C. LIU, B-D. 2003. A low-power precomputation-based fully parallel contentaddressable memory. Solid-State Circuits, IEEE Journal of. 654-662.

LIN, K-J. WU, C-W. 2000. A low-power cam design for lz data compression. Computers, IEEE Transactions on 49, 10 (October), 1139-1145.

LIU, S.C. F. W. and KUO, J. 2001. A novel low-voltage content-addressablememory (cam) cell with a fast tag-compare capability using partially depleted (pd) soi cmos dynamic-threshold (dtmos) techniques. Solid-State Circuits, IEEE Journal of. 712-716.

MAHONEY, P., SAVARIA, Y., BOIS, G., and PLANTE, P. 2005. Parallel hashing memories: an alternative to content addressable memories. IEEE-NEWCAS Conference, 2005. The 3rd In- ternational , 223-226.

MERIBOUT, M. A new scalable and reconfigurable architecture. Potentials, IEEE, 26-32. 2003.

NATARAJAN, A., JASINSKI, D., BURLESON, W., and TESSIER, R. 2003. A hybrid adiabatic content addressable memory for ultra low-power applications. In ACM Great Lakes Symposium on VLSI. 72-75.

PAGIAMTZIS, K. and SHEIKHOLESLAMI, A. 2004. A low-power content-addressable memory (CAM) using pipelined hierarchical search scheme. IEEE Journal of Solid- State Circuits 39, 9 (September), 1512-1519.

PENG, M. and AZGOMI, S. 2001. Content-addressable memory (cam) and its network applications. In International IC-Taipei Conference proceedings. ACM Transactions on Computational Logic, March 2001.

SAVARIA, Y., Conception et Vérification des Circuits VLSI. Août 1988. Presses Internationales Polytechnique.

SEDGEWICK, R. Algorithms in C, 3rd edition. Addison-Wesley. 2002.

SEZNEC, A. 1993. A case for two-way skewed-associative caches. In International Symposium on Computer Architecture. 169-178.

TARASSENKO, L. TOMBS, J.N. REYNOLDS, J.H. Neural network architectures for content-addressable memory. Radar and Signal Processing, IEEE Proceedings. 1991. Pages 33-39.

ZUKOWSKI, C. A. and WANG, S.-Y. 1997. Use of selective precharge for low-power on the match lines of content addressable memories. In MTDT 97: Proceedings of the 1997 IEEE International Workshop on Memory Technology, Design and Testing. IEEE Computer Society, Washington, DC, USA, 64-68.

ANNEXE I

PARALLEL HASHING MEMORIES: AN ALTERNATIVE TO CONTENT ADDRESSABLE MEMORIES. SOUMIS LE 31 MARS 2006 AU JOURNAL ACM TRANSACTIONS ON STORAGE.

I.1 abstract

This paper proposes a RAM based architecture that offers a functionality emulating the one of content addressable memories (CAMs), while offering lower power dissipation and requiring lower silicon area. The proposed architecture offers gracefully degradable behaviour when filling, which could best fit applications without a clear hard upper limit on the required number of stored items. An analytical model is elaborated so as to identify the architecture behaviour in domains that cannot be reached by simulations due to excessive processing time. Finally, several design optimizations are proposed in order to emphasize a particular metric at the expense of others in order to best fit the needs of a given application.

I.2 Introduction

Content Addressable Memories (CAMs) allow lookup operations to be executed in a single cycle. This feature enables many applications to sustain high data throughput in a deterministic manner.

Typical applications that leverage the CAM behaviour include "Ethernet address lookup, data compression, pattern-recognition, cache tags, high-bandwidth address

Tableau I.1 Static RAM and binary CAM technologies comparison. CMOS $0.18\mu m$ technology. 100 MHz operating frequency.

	CAM	RAM	CAM/RAM Factor
Cost $(mm^2/Mbit)$	19.2	6.75	2.8
Power Consumption Read operations $(mW/Mbit)$	410	80.9	5.1
Power Consumption Write Operations $(mW/Mbit)$	735	89.3	8.23

filtering, and fast lookup of routing, user privilege, security or encryption information on a packet-by-packet basis for high-performance switches, firewalls, bridges and routers." (Peng 2001)

Random Access Memory (RAM) technology does not offer single cycle lookup operations, yet it bears several key advantages over CAM technology. Typically, for a given storage capacity, it occupies smaller die area, it consumes less power, it possesses shorter access latency, it is offered in a wider variety of sizes and flavours. Also, it is more generic and widely available and enables to avoid the heavy licensing and royalty costs charged by some CAM vendors.

Table I.1 illustrates some of the previously mentioned advantages of RAM technology over CAM technology. This comparison is based on data taken from specification sheets generated by a memory compiler of a common but undisclosed proprietary source.

Because of all these reasons, many applications could benefit from an alternative solution to CAM technology which could offer improved price over performance or power consumption over performance ratios. This paper aims at offering such an alternative solution. It strives at emulating CAM behavior by using parallel static

RAM units, hence leveraging the benefits of using RAM technology so as to offer improved die area, power and energy consumption, as well as latency metrics.

Through simulation, analytical modeling and implementation, this paper will demonstrate clear advantages of the proposed architecture over standard CAM technology with respect to power consumption and die area metrics. It also shows that CAM's performance and operation throughput metrics can be matched and even surpassed for some applications.

While section 2 presents relevant work which has previously been accomplished in the field, section 3 describes the inner workings of the proposed architecture. Results obtained from simulations are presented in section 4 while section 5 concludes the paper.

I.3 Previous Work

Many research projects aimed at reducing the power consumption of CAM cells. Most of them ((Natarajan 2003), (Efthymiou 2002), (Pagiamtzis 2004), (Delgado-Frias 2005), (Liu 2001), (Lin 2003), (Zukowski 1997), (Lin 2000)) tried to achieve this goal by modifying the CAM unit cell that stores a single CAM bit. Several paths were explored, including the reorganization of the transistors so as to have as few transistors as possible discharging preloaded values to ground (Lin 2003).

Such novel CAM structures are costly to use in very large scale integrated circuit (VLSI) designs because typically, they must be supported by some CAM compiler. An approach that may prove easier and less expensive is to use RAM technology combined with some hashing scheme so combined to emulate a CAM behaviour.

Broder (Broder 2001) introduced a dynamic data structure that makes use of sev-

eral tables, each possessing a unique hash function. Insertion operations consist in trying to sequentially insert an element into every table until a successful attempt is made. In the event where all insertion attempts produce a collision, Broder proposes to rehash a subset of the previously stored elements in a way that would allow all of them to be contained.

Lim (Lim 2004; Lim 2002) extended this idea and proposed to implement it in hardware, using two RAM-based hash tables. Instead of rehashing, Lim added a small CAM unit which is used as an overflow table that stores elements that produced collisions in other tables. In a system composed of two RAM-based tables and one small CAM, when a lookup operation is made, all three units are accessed simultaneously. A priority encoder then takes care of routing the right information to the output.

Seznec (Seznec 1993) proposed a multi-bank cache configuration that could use more than one hash function. They were used to simultaneously map physical addresses to a distinct line in each bank. The performance gain is based on the high probability that addresses colliding in one bank would not collide in another.

The current paper extends these concepts in order to offer an improved alternative to CAM technology. It shows that with the use of hashing and RAM technology, the architecture proposed in (Mahoney 2005) can match and even surpass performances of CAM technology. The VLSI implementation shows that the architecture can reach these performances while still offering improved consumption and die area metrics.

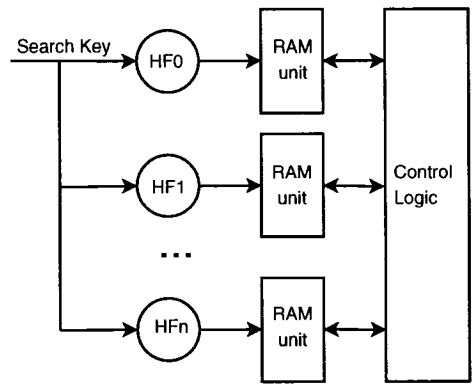

Figure I.1 Proposed hardware structure

I.4 Proposed Architecture

The proposed architecture is structured as sets of modules implementing multiple processing layers, each consisting of a hashing function, a static RAM unit and some combinational logic. The hashing functions translate the tag portion of the incoming data into an address of the RAM cell, as shown in Fig. I.1.

It is assumed, without loss of generality, that rows in memory contain at least 3 fields: the present field, the tag field and payload field. The present field occupies a single bit which is set if the row possesses valid data. The tag field contains the search key which will be compared to the supplied search key to signal a match. The payload field contains the sought-after data associated with the search key. For example, an 8 port Ethernet bridge could have a 48 bit wide tag field representing MAC addresses and a 3 bit wide payload field representing the port through which the associated MAC address can be reached. Considering the one bit present field, every row in memory would thus span 52 bits.

During a lookup operation, all layers execute a memory access to their RAM unit at the address specified by their respective hash key. The accessed data is then

compared to the supplied search key. Every layer passes on the comparison results along with the payload of the accessed row to the output logic unit. This unit gathers the information and signals a successful lookup operation if one of the layers signals a match. In this case, it also gates the appropriate payload on the output pins. If no match occurs, the output logic unit signals a failed lookup operation.

Insertion operations are analogous to their lookup counterpart. They can be split into 2 cases depending on whether or not the search key being inserted resides already in memory. If it is the case, the insertion operation consists in a payload update. If it is not, the insertion operation requires that a new row in memory be allocated to the data being inserted. For example, in an Ethernet bridge, executing an insertion operation with a non-resident search key might happen after receiving the initial frame from a given host. On the other hand, if this host has previously issued several packets, the inserted MAC address might already be present in memory. The inserted payload might be identical to the one being inserted, depending on whether or not the port through which the host can now be reached has changed since the last packet issued from the host was seen. In either case, it will be overwritten, and thus updated.

The insertion operation itself can be seen as including a lookup operation. The procedures are identical up to the point where the output logic unit gathers the match related information from the layers. This time, it sends write authorizations back to the layers. A positive authorization will be given to the layer signaling a match if any, thus causing a payload update. If no layer signals a match, the positive authorization will be given to the highest priority layer among those signaling an empty row. If no such layer exists, the insertion operation will be considered as having failed.

Because of the possibility of insertion failures, the designer will have to oversize the total architecture memory capacity in order to emulate a CAM unit of a given size. RAM's low area cost per bit compared to CAM's gives the designer the luxury of using only a fraction of the total memory capacity, while keeping the transistor count below the one of the emulated CAM unit. In this paper, we define the load factor as being the inverse of the oversizing factor. In addition to the load factor, directly linked to the total memory capacity, the number of layers is an application specific parameter chosen to obtain an acceptably small insertion failure rate.

Another configuration parameter resides in the choice of the hashing functions used. A good hashing function will uniformly spread the probability of associating any given search key between all possible hash key values, hence decorrelating any similitude in search key traffic. If there was a known ideal hashing function for all applications, such a function would be selected. As none is known, the choice of the hashing function can be seen as being application dependent. For instance, Jain (Jain 1992) has shown that bits generated by cyclic redundancy check (CRC) algorithms are close to those produced with ideal hashing functions. Thus such a hashing function would be a logical choice, yet CRC algorithms possess several parameters that can yield to very inadequate results if poorly chosen. This topic is complex, and even though further research would be justified, the issue is left for future work.

I.5 Performance Characterization

The main behavioral difference between a CAM unit and the proposed architecture is that the former will successfully keep inserting new entries until it reaches full capacity, upon which time any new insertion attempt will fail. On the other hand, the proposed architecture will slowly start failing before reaching full capacity.

Because most applications do not possess hard upper limits on the number of entries needing to be supported, this gracefully degradable behaviour, combined with the possibility of oversizing the capacity, while keeping complexity below that of a competitive CAM, can best fit the needs of many applications where designers currently see CAM technology as being mandatory. Two characteristics need to be analyzed in order to appropriately choose the target load factor and the number of layers for a given application: the insertion failure rate and its logical derivative, the probability of failure of any given insertion attempt.

A simulator was set up in order to more precisely illustrate the behaviour of these two characteristics. This simulator was used to characterize the failure probability as a function of the relevant design parameters. It is important to note that the search keys were generated by using the *rand* routine from Gnu's GLIBC. This situation is equivalent to having either uncorrelated traffic or using ideal hashing functions, both yielding perfectly distributed search keys. Also, the performance and behaviour of the proposed architecture depend on several key variables. We thus define L to be the number of layers, α, the load factor, C, the total architecture memory capacity and τ, the insertion failure rate.

The probability of failure of any given insertion attempt is defined as the probability that on a given insertion, in every layer, the row located at the address associated with the search key being inserted is occupied. This probability is bound to grow as the number of stored elements increase. It also depends on the number of layers, L, and on the total architecture capacity, C. Fig. I.2 illustrates the relationships between performance and these variables and parameters. The graph also characterizes the system behavior when the number of insertion attempts is larger than the capacity. Indeed, as insertion attempts may fail before the capacity is reached, some entries are empty when the number of insertion attempts reaches that capacity. Then further insertion attempts may be successful, even if the number of

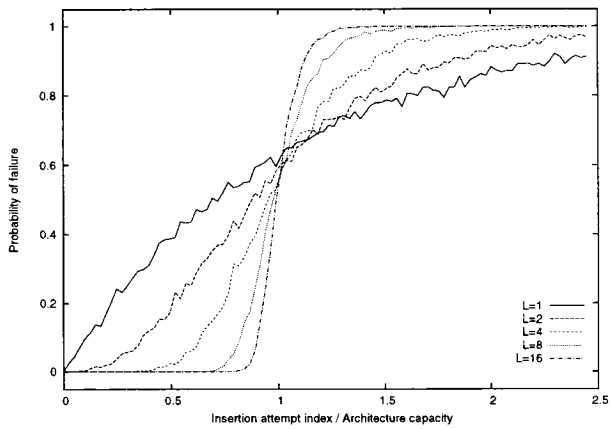

Figure I.2 The sigmoid behaviour of the probability of failure as the number of insertion attempts varies

insertion attempts is greater than capacity.

Fig. I.2 shows that as L increases, the sigmoid relation allowing to identify the probability of failure for any given number of insertion gets sharper. In other words, as the number of layers increases, the architecture behaves more and more like a CAM unit. It is also worthwhile to note that this statement holds true even as L approaches C where each RAM unit size gets closer to unity. In fact, a completely degenerated configuration where each RAM unit is of unity size, and where L equals C, can actually be seen as a non-optimized CAM unit implementation.

With this information, designers can fine tune the architecture configuration so as to obtain the sigmoid relation that best fits the application needs. For example, increasing the number of layers allows the application to execute a greater number of insertion attempts before reaching a given failure probability. As for the load factor, if the application possesses a soft upper-limit on the number of entries that it needs to store, decreasing the ratio between the number of insertion attempts and the capacity can be seen as shifting the sigmoid relation to the right. Fig. I.3 compares behaviours of a CAM unit and of the proposed architecture configured

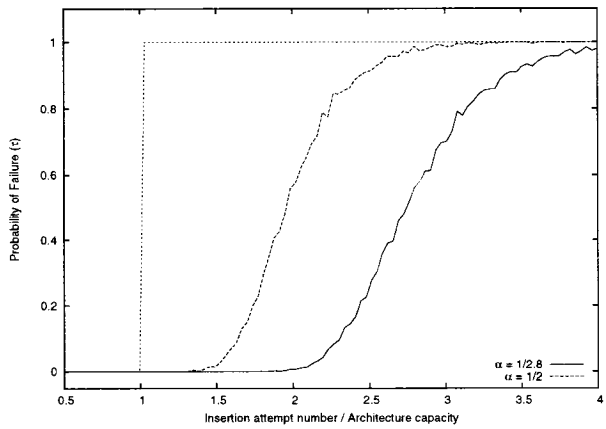

Figure I.3 Performance comparison with CAM at L=8 and $\alpha = 1/2.8$ (cost per bit equivalent) and $\alpha = 1/2$

with two different load factor parameters. Thus, as mentioned earlier, unlike the CAM unit, the proposed architecture can keep inserting new entries with a significant probability of success, even if the application goes beyond the target load factor. Note that a ratio of 2.8 in CAM to RAM capacities corresponds to the respective area per bit ratio listed in Table I.1.

The insertion failure rate for a given application is defined as being the number of failed insertion attempts encountered while the application was executing, over the number of distinct entries for which an insertion was attempted. This characteristic depends on both the number of layers and the load factor. As illustrated in Fig. I.4, the failure rate decreases in an exponential manner as the number of layers increases while keeping constant load factor values. Fig. I.5 presents that same behaviour from a different angle as the load factor varies for several values of number of layers.

Fig. I.6 combines L and α pairs yielding the same insertion failure rates. It allows to rapidly identifying all architecture configurations yielding to some predefined behavior. Fig. I.6 also suggests that failure rates of any magnitude can be obtained

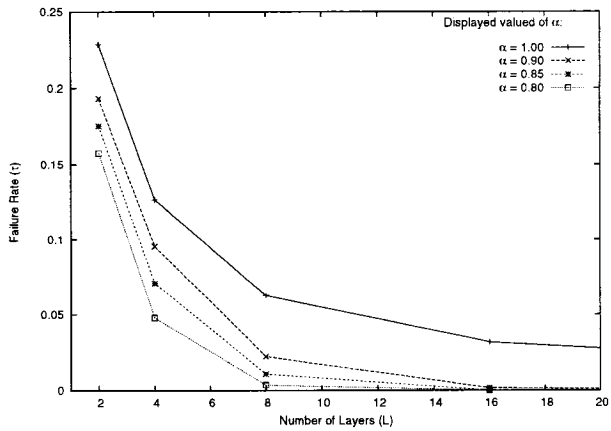

Figure I.4 τ vs L for several α values with constant total architecture capacity

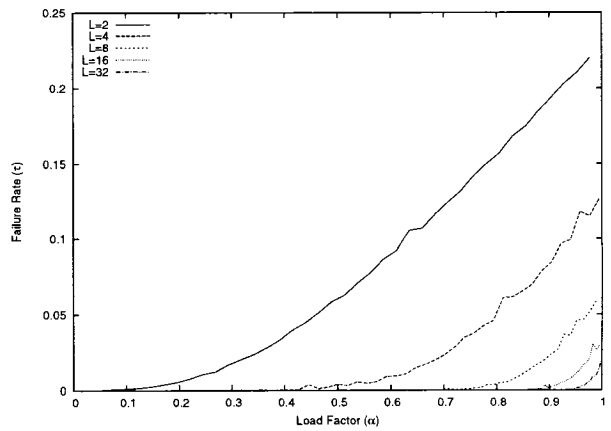

Figure I.5 τ vs α for several L values with constant total architecture capacity

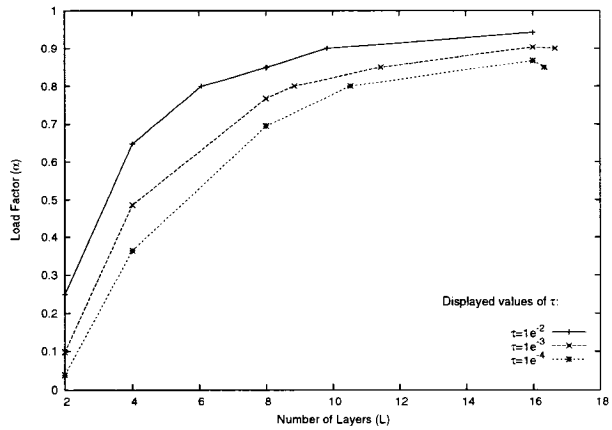

Figure I.6 L,α pairs yielding to constant τ values

by properly configuring the architecture. All applications can hence take advantage of the architecture by first identifying an insertion failure rate which it expects to support, and then choosing the L, α pair having the lowest production cost among those yielding the target behavior.

For example, a rate of 1 failure in every 10 years of execution time could be seen as being negligible for many applications. Such a value combined with a hypothetical 100 MHz operating frequency would translate into an overall failure rate of 3.171×10^{-16}. Unfortunately, prohibitive simulation times makes it difficult to characterize configurations producing rates of this order of magnitude. An analytical model is therefore needed to fill the gap.

I.6 Analytical Modeling of the Proposed Architecture

The insertion operations applied on the proposed architecture can be modeled with the balls and urns problem if ideal hash functions are used.

Let there be n balls and m urns. Let event E be the insertion of a ball into an urn

chosen in a perfectly random manner. The probability p that a given urn is chosen for a given insertion is:

$$p = \frac{1}{m} \tag{I.1}$$

If E is repeated for all the n balls, the probability that a given urn contains k balls is given by the probability mass function (pmf) equation of the binomial random variable:

$$P_k = \binom{n}{k} \left(\frac{1}{m}\right)^k \left(1 - \frac{1}{m}\right)^{n-k} \tag{I.2}$$

The pmf of the Poisson random variable is known to be a good approximation of the binomial's pmf when n is large and p is small. If we let $\beta = n/m$, equation I.2 can be rewritten as:

$$P_k = \frac{\beta^k}{k!} e^{-\beta} \tag{I.3}$$

Let the balls be the elements an application tries to insert into a hash table using an ideal hashing function, and let the urns be the rows of that table. In this case, β represents the load factor, and previously defined variable α can be used instead.

Let $O_{local}(\alpha)$ be the local occupancy relation, allowing one to obtain the occupancy rate of the hash table of a given layer. Since a given row will be occupied if the hash function assigns at least one element to it, $O(\alpha)$ is given by:

$$O_{local}(\alpha) = \frac{1}{m}\sum_{1}^{m} 1 - P_0 = 1 - P_0 = 1 - e^{-\alpha} \tag{I.4}$$

It is also worthwhile to note that the insertion of an element into a hash table possesses a probability of failure equal to the hash table's occupancy rate. Therefore, equation I.4 also defines the probability of failure for a single layered configuration as shown in Fig I.2 for $L = 1$.

In order to analytically express probability of failure and failure rate behaviours for all configurations, it is necessary to model multi layered units as systems composed of several single layered units. For this purpose, let $T = C/L$ be the number of rows in the RAM unit of each layer, O_i, α_i, and n_i respectively be the occupancy rate, the load factor and the number of insertion attempts of the i^{th} layer. We therefore have:

$$O_i = O_{local}(\alpha_i) \tag{I.5}$$

$$\alpha_i = N_i/T \tag{I.6}$$

When an insertion attempt made on a given layer fails, the attempt percolates to the next one. It can therefore be said that the number of insertion attempts made in a given layer is equal to the number of insertion attempts that failed on the previous one, as Fig. I.7 shows. Conversely, the number of insertion attempts made in a given layer is equal to the total number of insertion attempts made in the previous layer minus the number that succeeded.

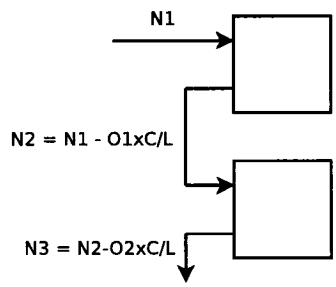

Figure I.7 If the insertion attempt of an element in a layer causes a collision, it is said to percolate to to the next layer.

$$n_i = N_{i-1} - O_{i-1} \times T \tag{I.7}$$

Relations I.7, I.5 and I.6 allow to express the global occupancy rate of a multi layered configuration as the sum of the occupancies of all the layers.

$$O_{global}(\alpha) = \frac{1}{L} \sum_{i=1}^{L} O_i \tag{I.8}$$

The failure rate can be obtained by normalizing the part of the load factor that corresponds to failed insertion attempts by the load factor itself.

$$\tau = \frac{\alpha - O_{global}(\alpha)}{\alpha} \tag{I.9}$$

The probability of failure depends on the state of the architecture at the moment when the insertion is made. In multi layered configurations, this value represents the probability that the inserted element collides in every single layer. It is thus given by the product of the individual occupancy rates of every layer.

Tableau I.2 Six layers are necessary to obtain a value of $\tau = 10^{-16}$. N_1=4000.

Layer	α_i	O_i	O_{global}	τ	n_{i+1}
1	1.500	0.7768	0.1294	0.4820	1928.3
2	0.7231	0.5147	0.2152	0.1389	555.6
3	0.2083	0.1880	0.2466	0.0135	54.066
4	0.0202	0.0200	0.2499	1.361E-4	0.5444
5	2.041E-4	2.041E-4	0.2499	1.389E-8	5.556E-5
6	2.083E-8	2.083E-8	0.2499	2.220E-16	5.921E-13

$$P_{failure} = \prod O_i \tag{I.10}$$

Relations I.9 and I.10 allow to analytically define the behaviour of the architecture. They not only validate our simulation results as we were able to analytically reproduce Figs. I.4 and I.5, but they also allow quantifying the behaviour in domains located beyond the practical limits imposed by simulation times that can become excessive.

For example, an application might need to store data up to a soft limit of 4000 elements. Its designer might judge a rate of 1 failure in every 10 years of execution time as being negligible, thus yielding to a τ value of 3.171×10^{-16}, as previously mentioned. Table I.2 shows that a $\alpha = 0.25, L = 6$ configuration offers the targeted performance. The design would thus have a total capacity of $C = 4000/0.25 = 16000$ elements divided into 6 RAM units of 16000/6 rows each.

The analytical model allows to easily plot the surface illustrating the behaviour of τ in regards to both parameters L and α, as shown in Fig. I.8. It also allows extending the characterized domain to failure rates beyond the limits feasible with affordable simulation times. Fig I.9 shows L, α pairs yielding to same failure rates. It is worthwhile to note that these relations are in fact iso-performance curves extracted from the surface of Fig. I.8.

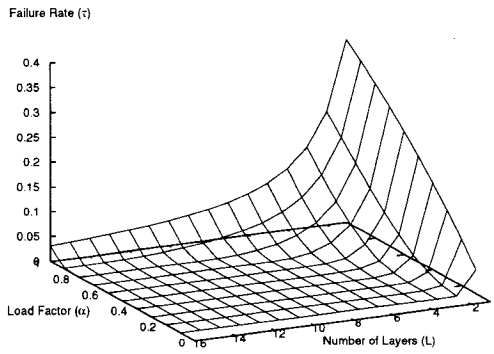

Figure I.8 Joined τ vs R and τ vs α relations

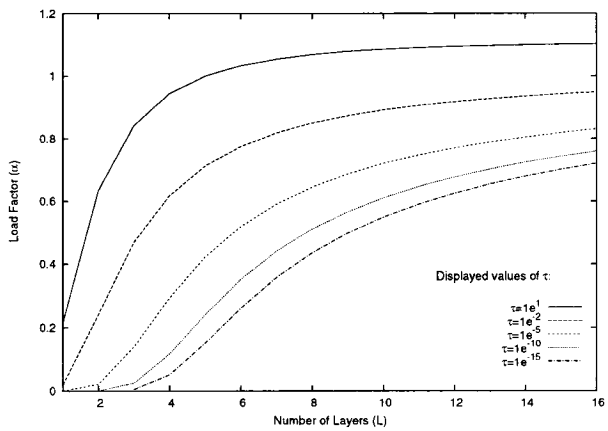

Figure I.9 α/L relations yielding to same τ

I.7 Proposed Methodology

One problem that has yet to be approached resides in the choice of the L, α pair to use for a given application. One possible methodology consists in identifying the target behaviour and then choosing the L, α pair leading to the lowest production cost among those yielding the desired behaviour.

One technique suitable for the graphically inclined designer consists in first elaborating a cost model through a surface identifying the die area or cost in dollars associated with every L, α pair. Next, the technique consists in finding the parametric relation of the 3D curve resulting from the projection of the α vs L with constant target τ relation as found in Fig. I.9 on the cost surface. This parametric relation defines the cost through a single parameter t from which both L and α can be obtained. The last step is to find the t value for which the lowest cost is observed. The resulting L and α values can then be considered as the best possible solution.

For example, let the cost model be $L^2+(1/\alpha)^2$ as shown in Fig. I.10, and the target failure rate be 10^{-15}. This cost model is only proposed for illustrative purposes. In order to make the projection, the α vs L with constant target $\tau = 10^{-15}$ relation needs to be modeled. Fig. I.11 shows that least squares fitting finds relation I.11 with parameters $a = 9.60$, $b = 10.21$ and $c = 0.051$ to be the best fit. This relation projected on the cost surface model yields the relation shown in Fig. I.12 and is minimal at integer value $t = 6$. Parameters $L = 6$ and $\alpha = 0.286$ are obtained by applying relations I.12 and I.13. Based on the cost surface model of Fig. I.10, these values yield a cost of $6^2 + (1/6)^2 = 48.2$.

$$f(x) = a - (\frac{b}{x^c}) \tag{I.11}$$

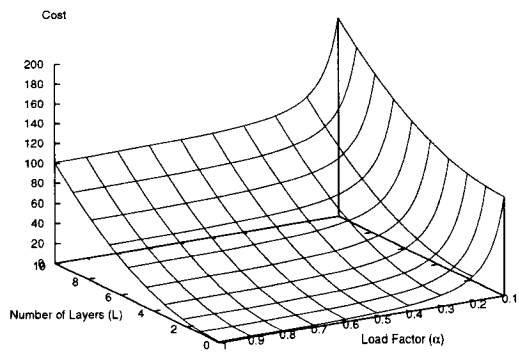

Figure I.10 Cost Model

$$L(t) = t \tag{I.12}$$

$$\alpha(t) = a - (\frac{b}{t^c}) \tag{I.13}$$

I.8 Hardware Implementation and Metric Comparison

A VLSI implementation of the proposed architecture has been realized in order to obtain a more thorough understanding of how it could compare to commercial CAMs.

The original implementation consists in synchronous single port RAM units glued together with combinational logic described with register transfer level VHDL. It is a multi-cycle design where read operations consume one cycle and where write operations consume two. This implementation is dominated by RAMs and requires minimal glue logic.

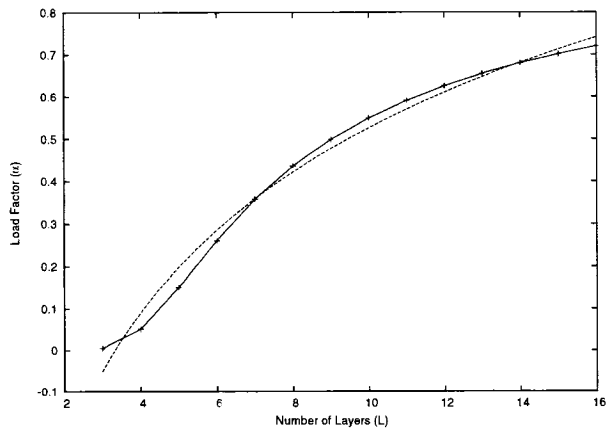

Figure I.11 Model of α vs L relation for $\tau = 10^{-15}$.

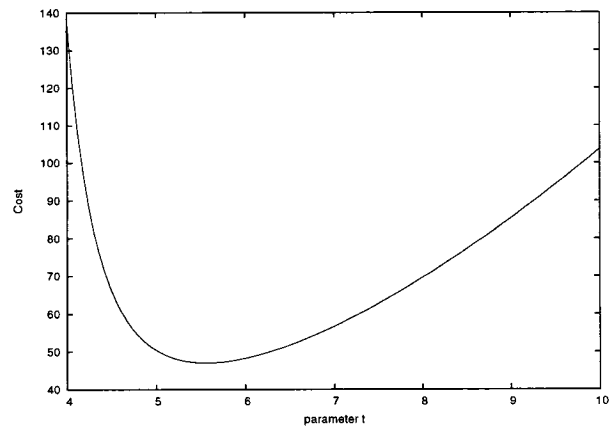

Figure I.12 Minimal cost configuration for $\tau = 10^{-15}$.

Tableau I.3 Bringing the clock period from over 25ns down to under 5ns does not affect significantly the portion of the total area occupied by the RAMs. 180nm technology.

Clock Period (ns)	Total Area (sq μm)	Combinational Area (sq μm)	RAM Portion
26.98	1,699,135	22,955	98.65%
9.48	1,699,518	23,338	98.63%
4.35	1,704,266	28,086	98.35%
4.27	1,707,043	30,863	98.19%
4.24	1,708,629	32,449	98.10%

Table I.3 presents area consumption data for a 4-layer implementation with 1024x52 RAM cells. It shows that whether the synthesis tries to minimize either clock period or die area, glue logic area remains small compared to the one consumed by the RAM cells. The proposed architecture hence greatly benefits from the high density of RAM technology while emulating CAM behaviour.

Table I.3 also shows that a clock frequency greater than 200 MHz can be reached. This value could be easily increased by using faster RAM cells, as 2.8 ns of the 4.24ns clock period is consumed by the RAM access time.

It is worthwhile noting that in order to obtain easily reproducible results, the implementation uses several standard modules from Synopsys' *DesignWare* library. One of them is the combinational CRC which is used for the hash function blocks. Synopsys' *Design Compiler* was also used to obtain both latency and die area estimates. As for power consumption of the logic, a tool developed by our industrial partner was used to produce the listed estimates. These estimates are based on the number of gates and on technology specific constraints.

Table I.4 presents a comparison of the design with a commercial CAM cell. It shows that the parallel hashing memories consume 2.4 less die area and 4.7 less

Tableau I.4 Comparisons of design with commercial CAMs. All values are for 180nm implementations running at 200MHz with a RAM access time of 2.8 ns.

	Parallel Hashing	Portion for RAM	CAM	CAM/PH Factor
Cost $(mm^2/Mbit)$	8.02	98.10%	19.2	2.4
Power Consumption $(mW/Mbit)$	309	68.8%	1454	4.7

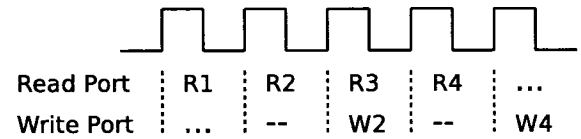

Figure I.13 Dual Port RAM design optimisation. Operations 2 and 4 represent insertions.

power than commercial CAM cells. As expected, these values are slightly lower than the ones presented in table I.1, yet they still offer an appealing alternative to CAM technology free of any royalty fee.

I.9 Proposed Optimisations

This section proposes three possible variations which can be brought to the design in order to optimize either power consumption, throughput or latency.

The first of them consists in using dual port RAM cells in order to support both lookup and insertion operations in a single cycle. Eliminating multi-cycle operations greatly simplifies interactions with the design, but on the other hand, it also considerably increases the die area consumed by the cell. In fact, dual port RAM cells can occupy close to twice the area of their single port counterpart of equal size.

With dual port RAM cells, one port can be dedicated solely for read operations, while the other takes care of write operations. It is hence possible to have the writing portion of insertion operations and the read portion of the following operation overlap each other, whether it is a lookup or an insertion. The writing portion of insertion operations is completely hidden to the user. The cell thus appears as completing every operation within a single cycle, as shown in Fig. I.13.

Since dual port RAM cells do not support simultaneous access to a same memory location, special wrapping logic is necessary to ensure correct behaviour when two or more consecutive operations are executed on the same RAM address. It basically consists in bypassing the read operation by presenting the data being written to the port wrapper where the read operation occurs.

The first consequence of the described design variation would be to bring the latency down to a single cycle. As for die area, since RAM cells represent around 98% of the initial implementation's consumption, the area could double. Finally, based on the 15% increase in power consumption of dual port RAMs over their single port counterparts and on the addition of the bypass logic, the increase in power consumption can be estimated as being close to 15%.

The second design variation offers an increase in throughput. It consists in pipelining the architecture so as to allow a greater number of operations to be completed in a given time unit. The pipeline could possess 3 stages. The first stage would essentially allow the hashing function unit to translate the search keys into RAM addresses. The second would consist in a RAM access. The third and last one would allow post-processing, such as gating the payload field to the output pins, to take place. Since the longest atomic action of the initial design is the RAM memory access, the clock period of the pipelined design would be defined by the RAM latency. Available RAM specification sheets define this value to be 2.8 ns

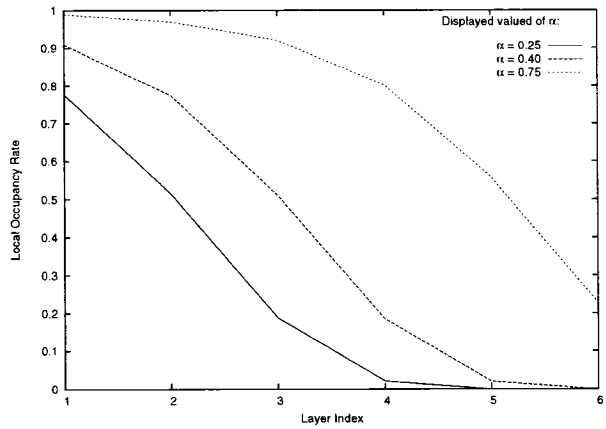

Figure I.14 The occupancy rate of the individual layers decreases as the layer index increases.

for a 180nm CMOS technology, hence the pipelined design could run at a 350MHz clock frequency. This design variation, combined with the dual-port RAM presented earlier, could allow the presented architecture to fully complete 350×10^6 operations per second, whether they are lookup or insertion operations.

The third and last design variation offers a decrease in power consumption at the expense of a greater average latency.

Because of the percolation process described in section III, the highest indexed layers fill up faster than the lower indexed ones. In fact, relation I.7 shows that a given layer cannot possess a number of elements greater than any higher indexed layer. In other words, $N_i \leq N_j$ if $i < j$. Based on this rule, it is safe to assume that read operations executed on a given layer possess a greater probability of success than the ones executed on any of the lower indexed layers. This holds true particularly if the load factor is small, as illustrated in Fig. I.14.

This design variation aims at taking advantage of this characteristic by sequentially accessing groups of layers until a match is found. It offers a decrease in power

consumption as lookup operations will not require accessing every RAM cell at all times. As for the increase of latency, its average value depends on both the group size and the load factor, while its worst case value equals the number of groups in the design.

For example, let us reuse the 6 layer, 0.25 load factor configuration of Table I.2 and apply the current design variation by grouping the layers in groups of 2. Based on the local occupancy rate of each layer, the first group can be shown as possessing 86.1% of all stored elements, while the last two would possess respectively 13.9% and 0.082%. A simple weighted sum leads one to identify the average latency of lookup operations executed on stored elements as being of 1.385 cycles. In a similar way, the decrease in RAM cell power consumption can be evaluated as being close to 62%. Even if neglecting power consumption of the additional logic required for this design variation, it is clear that many applications could benefit from this kind of tradeoff.

I.10 Conclusions

An architecture able to emulate CAM technology units was presented. It offers notable improvements on power consumption and die area metrics in addition to avoiding the high licensing fees charged by some CAM vendors. Because of its gracefully degradable performances, it can also best fit the needs of applications that do not possess hard upper limits on the number of items to store and whose designers currently see CAM technology as being their only choice.

The performances of the presented architecture obtained in simulations were backed by an analytical model which allows to identify the configurations with low failure rates. A method to identify optimal configuration parameters was also presented,

hence handing to the reader the tools he needs in order to start designing efficiently.

Several design variations were also presented. The first consists in using dual port RAMs so as to have both lookup and insertion operations completed in a single cycle. The second pipelines the design in three stages in order to boost the throughput, and to bring the clock period down to the value of the RAM access time. The third and last one proposes to sequentially compare groups of layers until the sought after item is found. Unaccessed RAM cells being disabled, power consumption gets reduced at the expense of an increased average latency.

Both simulation results and analytically obtained performance metrics assume perfect hashing functions. Even though CRC algorithms are considered as being the best available hashing functions (Jain 1992), not much work has been done on the topic. Several papers offer analysis on the behaviour of CRC algorithms with respect to their capability to detect and repair corrupted data, yet none of them discuss their effectiveness to offer well dispersed hash keys. This kind of analysis might be necessary before bringing the proposed architecture in the field.

Because of its simple design and of the widely spread use of the required tools, the presented architecture offers a very appealing alternative to CAM technology. Its performance metrics are also bound to improve over time, as CAM technology is not subject to the very intense commercial competition found in the RAM market and thus does not evolve as fast. Many applications like address lookups, cache tags and pattern recognition could greatly benefit from using it.

I.11 acknowledgments

The authors of this paper would like to acknowledge financial contributions from Micronet R&D, the Canadian Microelectronics Corporation and PMC-Sierra, Inc.

They would also like to thank Benoit Côté, Martin Bisson and Normand Bélanger for their respective contribution to the project

RÉFÉRENCES

BRODER, A. and KARLIN, A. 1990. Multilevel adaptive hashing. In Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. ACM, San Francisco, California, United States, 43-53.

LIN, C-S. CHANG, J-C. LIU, B-D. 2003. A low-power precomputation-based fully parallel content-addressable memory. Solid-State Circuits, IEEE Journal of. 654-662.

EFTHYMIOU, A. and GARSIDE, J.D. 2002. An adaptive serial-parallel cam architecture for low-power cache blocks. In Proceedings of the 2002 international symposium on Low power electronics and design. 136-141.

LIM, H. SEO, J-H. JUNG, Y-J. 2003. High speed ip address lookup architecture using hashing. IEEE Communications Letters. 502-504.

JAIN, R. 1992. A comparison of hashing schemes for address lookup in computer networks. Communications, IEEE Transactions on. 1570-1573.

DELGADO-FRIAS, J.G. NYATHI, J. TATAPUDI, S.B. 2005. Decoupled dynamic ternary content addressable memories. Circuits and Systems, IEEE Transactions on 52, 10 (October), 2139-2147.

LIN, K-J. WU, C-W. 2000. A low-power cam design for lz data compression. Computers, IEEE Transactions on 49, 10 (October), 1139-1145.

LIM, H. and JUNG, Y. 2004. A parallel multiple hashing architecture for ip address lookup. High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on, 91-95.

MAHONEY, P., SAVARIA, Y., BOIS, G., and PLANTE, P. 2005. Parallel hashing memories: an alternative to content addressable memories. IEEE-NEWCAS Conference, 2005. The 3rd In- ternational , 223-226.

NATARAJAN, A., JASINSKI, D., BURLESON, W., and TESSIER, R. 2003. A hybrid adiabatic content addressable memory for ultra low-power applications. In ACM Great Lakes Symposium on VLSI. 72-75.

PAGIAMTZIS, K. and SHEIKHOLESLAMI, A. 2004. A low-power content-addressable memory (CAM) using pipelined hierarchical search scheme. IEEE Journal of Solid-State Circuits 39, 9 (September), 1512-1519.

PENG, M. and AZGOMI, S. 2001. Content-addressable memory (cam) and its network applications. In International IC-Taipei Conference proceedings. ACM Transactions on Computational Logic, March 2001.

LIU, S.C. WU, F.A. KUO, J.B. 2001. A novel low-voltage content-addressable-memory (cam) cell with a fast tag-compare capability using partially depleted (pd) soi cmos dynamic-threshold (dtmos) techniques. Solid-State Circuits, IEEE Journal of. 712-716.

SEZNEC, A. 1993. A case for two-way skewed-associative caches. In International Symposium on Computer Architecture. 169-178.

ZUKOWSKI, C. A. and WANG, S.-Y. 1997. Use of selective precharge for low-power on the match lines of content addressable memories. In MTDT 97: Proceedings of the 1997 IEEE International Workshop on Memory Technology, Design and Testing. IEEE Computer Society, Washington, DC, USA, 64-68.

ANNEXE II

PARALLEL HASHING MEMORIES: AN ALTERNATIVE TO CONTENT ADDRESSABLE MEMORIES. PRÉSENTÉ À LA CONFÉRENCE IEEE NEWCAS 2005.

II.1 abstract

Content addressable memories, or CAMs, are commonly used in applications requiring high speed access to data sets. This technology allows data items to be accessed in constant time based on content rather than on address. Unfortunately, this technology has several drawbacks: it occupies more die area per bit, costs more, dissipates more power, and has a higher latency. This article proposes an alternative to CAM technology based on a parallel hashing architecture. Simulations show that CAM performances can be matched and even surpassed while reducing cost and power consumption. The tradeoffs that exist between performance and cost are explored in the paper.

Index Terms: content addressable memory (CAM), parallel hashing, address lookup, cache tags.

II.2 Introduction

Content addressable memory technology is widely used in high throughput applications requiring frequent memory lookup operations. It allows data to be accessed by its content rather than by its address. In a constant number of clock cycles,

CAM units will compare a user supplied key to the tag field of all the rows in memory and will route information that matches back to the user. One or more matches may occur if "don't cares" are allowed or if several items possess a common tag field.

According to Peng (Peng 2001), "CAM technology is ideally suited for several applications, including Ethernet address lookup, data compression, pattern-recognition, cache tags, high-bandwidth address filtering, and fast lookup of routing, user privilege, security or encryption information on a packet-by-packet basis for high-performance switches, firewalls, bridges and routers."

CAM technology presents major advantages over standard RAM, yet it also has shortcomings. Typically, it is more costly, it consumes more power, it has a limited size, a high latency, and evolves slower, as it is not subject to the very intense commercial competition found in the RAM market. For these reasons, there is a need for an alternative technology that matches CAM functionality with improved performance or improved price/performance or power/performance ratios.

This paper proposes a memory architecture that aims at being such an alternative. It emulates CAM functionality with parallel RAM based hash tables. It hence leverages all of the above mentioned advantages of RAMs over CAMs. Unlike CAMs that work flawlessly until it reaches full capacity, at which point it stops accepting new data, the probability of failure of the proposed memory architecture starts rising before it reaches full capacity, offering gracefully degradable performances as the number of stored elements increases. As it will be shown, because a RAM bit size is smaller than a CAM bit size, oversizing the proposed architecture can produce a RAM based memory module that outperforms CAMs on all aspects for most applications.

The rest of this paper is organized as follows. Section II describes relevant prior work on the topic. Section III presents a brief comparison of CAM and RAM technologies while Section IV describes the proposed architecture. A summary of key simulation results is presented in Section V and these results are analyzed in Section VI. Finally, our main conclusions are summarized in Section VII.

II.3 Previous Schemes

There are essentially two schemes on which the proposed architecture is based: multilevel adaptive hashing and skewed associative caches.

Broder (Broder 1990) introduced a dynamic data structure which makes use of several tables, each possessing a unique hash function. Insertion operations consist in trying to sequentially insert an element into every table until a succesfull attempt is made. In the event where all insertion attempts produce a collision, Broder proposes to rehash a subset of the previously stored elements in a way that would allow all of them to be contained.

Lim (Lim 2003) picked up this idea and proposed to implement it in hardware, using two RAM-based hash tables. Instead of rehashing, Lim adds a small CAM unit which is used as an overflow table that stores elements that produce collisions in both tables. When a lookup operation is made, all three units are accessed simultaneously. A priority encoder then takes care of routing the right information to the output.

Seznec (Seznec 1993) proposed a multi-bank cache configuration which made use of more than one hash functions. They were used to simultaneously map physical addresses to a distinct line in each bank. The performance gain is based on the high probability that addresses that collide in one bank would not collide in another.

The current paper extends these concepts in order to offer an alternative to CAM technology. It proposes to implement Broder's scheme as Lim and Seznec did. Instead of using rehashing, which appears to be costly and time consuming when implemented in hardware, or using a CAM unit as an overflow table, this paper proposes configuring the structure as a function of application requirements in order to have a negligible insertion failure rate at execution time.

II.4 Binary CAM vs RAM

Tableau II.1 Static RAM and binary CAM technologies comparison. CMOS $0.13\mu m$ technology. 100 MHz operating frequency.

	CAM	RAM	Factor
Cost $(mm^2/Mbit)$	15.5	3.6	4.3
Power Consumption Typical Read Operation $(mW/Mbit)$	129	42	3.1
Power Consumption Typical Write Operation $(mW/Mbit)$	250	55	4.5
Latency (ns)	3.7	3.0	1.2

Table II.1 summarizes a comparison between static RAM and binary CAM technologies. This comparison is based on data taken from specification sheets generated by a memory compiler of a common but undisclosed proprietary source.

Comparing the cost per bit of both technologies reveals only part of the story, as tools and licensing fees may represent a major part of the total cost of using CAMs in an application specific integrated circuit (ASIC). Yet only considering the cost per bit metric, CAM is more expensive by a factor larger than 4. This factor is bound to increase because, according to Peng (Peng 2001), "RAM technology

is driven by many applications, particularly computers and consumer electronics products, and hence the cost per bit is continuously decreasing. CAM technology is considered specialized and only a modest increase in bit capacity and decrease in cost is expected."

Comparing power consumption is very tricky, as CAM's consumption greatly varies depending on the operating frequency, the key content, and the match rate. Based on what could be defined as standard use for most applications, for a given memory size, the CAM consumes more power by a factor larger than 3 for read operations and larger than 4 for write operations.

II.5 Proposed Architecture

The proposed architecture consists in a certain number of layers, each composed of a hash function unit and a RAM-based hash table, as illustrated in Fig. II.1.

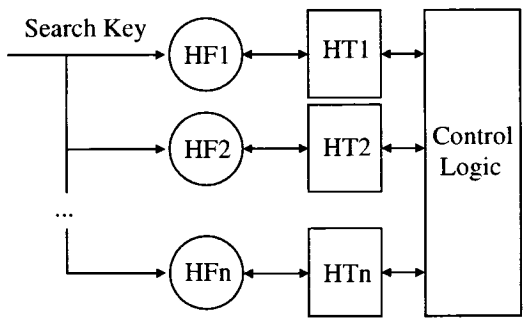

Figure II.1 Proposed hardware structure

A hash function unit converts a search key into a hash key in a deterministic manner. The generated hash key is then used as an address to execute read or write operations into a hash table. Collisions do occur, but good hash functions will keep their count low by breaking correlations in hash key traffic and, when

successful, will thus yield highly randomized hash keys.

As illustrated in Fig. II.2, each hash table entry contains three fields: the present bit, the tag and the payload.

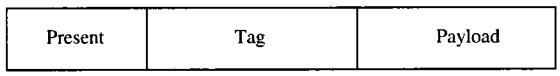

Figure II.2 Hash table entry layout

When an insertion attempt is made, every hash function generates a hash key and retrieves the corresponding table entry. If the present bit is set, the retrieved entry is then compared to the inserted key. The result is forwarded to the control logic unit. This unit will authorize the write operation to the layer where a match has occurred, therefore avoiding duplicate entries. If no match is indicated, the unit will then authorize the write operation to the highest leveled layer among those signaling an empty entry. If no layer signals an empty entry, the write operation is considered as having failed.

The process is analogous for lookup operations. When the search key is presented to the hash function units, the hash keys are generated and the corresponding table entries are retrieved and compared to the search key. The payload portion of a match-signaling layer is then gated to the output. If no layer signals a match, the lookup operation is considered as having failed.

This architecture is similar to those described in section I. The main difference is that it does not support rehashing, nor does it require an overflow CAM table. It is designed to be configurable in terms of the number of layers and of the target load factor, so as to be an appropriate alternative to CAM technology for most applications.

II.6 Simulation Results

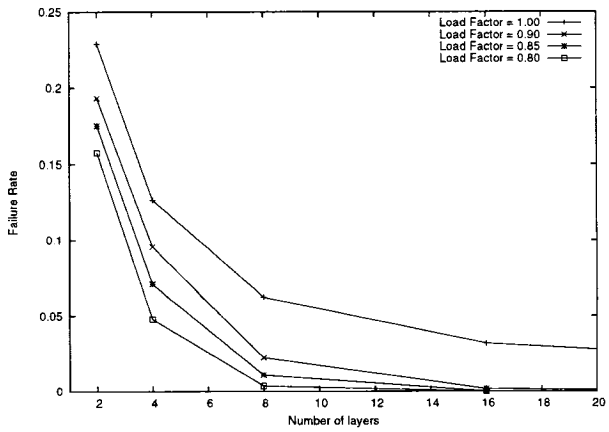

Figure II.3 τ vs L for several α values with constant total architecture capacity

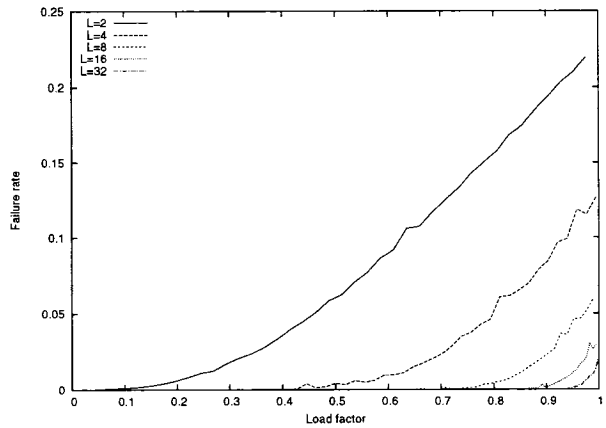

Figure II.4 τ vs α for several L values with constant total architecture capacity

An issue that must be considered with the proposed architecture is the possibility that one or more insertion attempts result in a failure. A failed insertion attempt occurs when all the individual attempts made on the hash tables produce a collision. There is therefore a critical need to evaluate the architecture performance under this metric.

The insertion failure rate, τ, depends on several configuration parameters: the hash functions used, the total architecture capacity C, the number of layers L and the load factor α. The architecture capacity is given by L multiplied by the size of individual memories. The load factor α is the fraction of the architecture capacity used up at a certain point. Adding more layers for a same architecture capacity thus reduces the size of each individual memory accordingly.

A simulator was created to evaluate the influence of the last two parameters on the insertion failure rate. The simulations were done with perfectly randomized search keys. This approaches the situation where perfect hash function units are used, as hashed data are already uncorrelated, hence allowing hash keys to be evenly distributed over a certain period of time, no matter which hash functions are used.

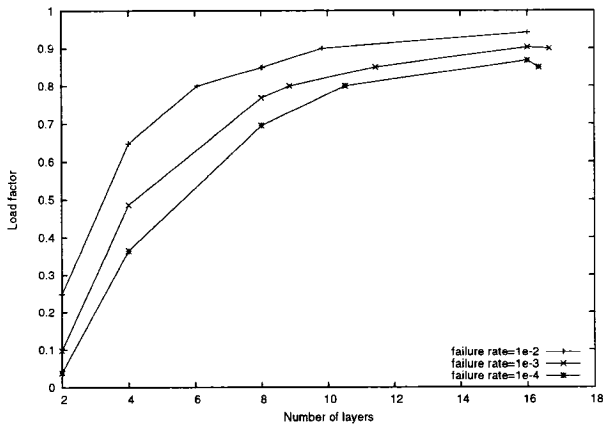

Figure II.5 L,α pairs yielding to constant τ values

The simulations first allowed us to visualize the relation between τ and L. Fig. II.3 shows the e^{-kL} behavior of τ as the number of layers varies. Simulations also have enabled us to see the relation between τ and α. Fig. II.4 illustrates what seems to be another exponential behavior of τ as the load factor varies. These simulations were done with a constant C value in order to compare the multiple possible configurations with a common total architecture capacity. Note that the

glitches observed in Fig. II.4 are probably due to the finite number of simulation runs from which those results were obtained.

These exponential relations led us to believe that many L, α pairs can yield the same τ value. This is confirmed by results in Fig. II.5 which shows L, α pairs yielding to τ values 10^{-2}, 10^{-3} and 10^{-4}. Lower τ values can be easily obtained, but prohibitive simulation time makes it difficult to characterize the L, α pairs producing them.

Another metric which helps evaluate the architecture's performance is the probability of failure of any given insertion attempt. Simulations show how this value increases with the number of past insertion attempts. Fig. II.6 shows that the sigmoid function representing the probability of failure for all insertion attempts gets sharper as the number of layers increases.

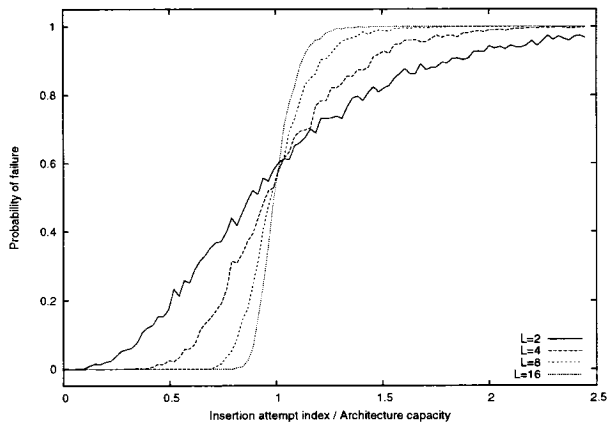

Figure II.6 Probability of failure vs $\frac{InsertionAttemptIndex}{TotalCapacity}$ for L=2,4,8,16

II.7 Results Analysis

The increasing sharpness in the sigmoid function relation shown in Fig. II.6 is a logical consequence of the architecture core behavior. As the number of layers increases, while keeping a constant total capacity, the architecture increasingly behaves as a CAM unit. In fact, in an extreme configuration, where the number of layers is equal to the total architecture capacity, each having a hash table of unity size, the architecture can be seen as CAM implementation.

The sigmoid function defining the probability of failure behavior for any given insertion attempt can be a significant advantage of the proposed architecture over CAM technology. In a CAM unit, all of the insertion attempts made beyond the claimed supported number of entries will fail, whereas it would succeed with a high probability in an oversized configuration of the proposed architecture, with a probability of failure slowly rising with the load factor.

This situation is illustrated in Fig. II.7. The left-most relation shows the behavior of a CAM unit whose probability of failure jumps to 100% as the number of insertion attempts reaches the claimed supported number of entries ($\alpha = 1$). The right-most relation presents a $L = 8, \alpha = 1/4.3$ configured architecture. Data in table II.1 allows one to view this configuration as a CAM alternative with equivalent cost-per-bit if the control logic cost is neglected. Fig. II.5 shows that this configuration will reach a failure rate of 10^{-4} only after having made a number of insertion attempts larger than 3 times the claimed number of supported entries. Considering the exponential relation shown in Fig. II.4, it is safe to say that normal use of the architecture ($\alpha \leq 1$) can have a negligible failure rate for most applications.

Even if control logic is not neglected, it is possible to obtain superior performances at a lower cost. Fig. II.7 also shows hows a $L = 8$ architecture, configured with

$\alpha = 1/2$ would compare to a CAM. It offers features similar to the $\alpha = 1/4.3$ configuration, at a nominal cost lower than CAM technology, since it is safe to say that even when considering control logic, CAM technology would have a cost-per-bit larger than twice the one of our architecture.

The simulation results lead us to believe that a failure rate as small as desired can be obtained by configuring the architecture properly. The designer simply needs to identify what failure rate can be considered negligible or acceptable for the application, and then find the corresponding L, α pair having the lowest production cost.

The results reported here have been generated with perfectly distributed search keys. One could argue that real-life applications rarely face such situations, and that the simulation results would not stand in the field. This mostly depends on the hash functions used. Good hash function will successfully uncorrelate any similitude in search key traffic. Jain (Jain 1992) has shown that bits generated by cyclic redundancy check (CRC) algorithms are close to those produced with ideal hash functions. Performances in the field can hence be expected to be close to simulation results with a certain degree of confidence if such hash functions are used.

II.8 Conclusion

A RAM-based parallel hashing memory architecture which emulates the general functionality of a CAM was proposed. It not only provides superior performances compared to CAM units for most applications, but it also offers three distinct additional advantages: lower silicon area cost per bit, lower tools and licensing cost and lower power consumption.

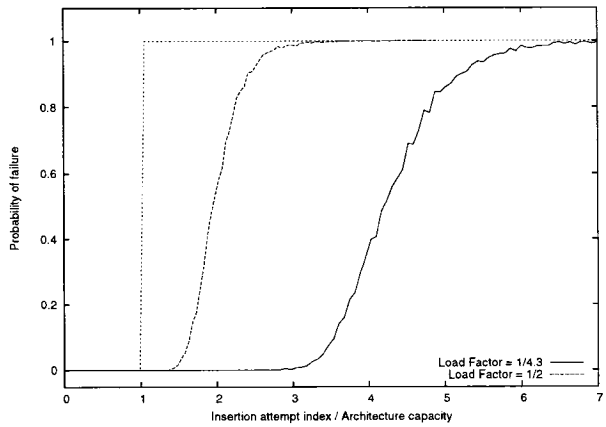

Figure II.7 Performance comparison with CAM at L=8 and $\alpha = 1/4.3$ (cost per bit equivalent) and $\alpha = 1/2$

It therefore represents a very interesting alternative to CAM technology that could be leveraged in many possible applications of CAMs. Typical applications include address lookup operations, cache tags and encryption. It can be configured to a wide range of applications and desired levels of performance.

The architecture proposed in this paper is described in its simplest form. Several features could be added, such as support for multiple match, configurable word-width, aging mechanism, database partitioning, search and delete operations and wildcards.

II.9 Acknowledgments

The authors of this paper would like to acknowledge financial contributions from Micronet R&D, the Canadian Microelectronics Corporation and PMC-Sierra, Inc. They would also like to thank Hung Tien Bui for his contribution.

RÉFÉRENCES

BRODER, A. and KARLIN, A. 1990. Multilevel adaptive hashing. In Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. ACM, San Francisco, California, United States, 43-53.

SEZNEC, A. 1993. A case for two-way skewed-associative caches. In International Symposium on Computer Architecture. 169-178.

LIM, H. SEO, J-H. JUNG, Y-J. 2003. High speed ip address lookup architecture using hashing. IEEE Communications Letters. 502-504.

JAIN, R. 1992. A comparison of hashing schemes for address lookup in computer networks. Communications, IEEE Transactions on. 1570-1573.

PENG, M. and AZGOMI, S. 2001. Content-addressable memory (cam) and its network applications. In International IC-Taipei Conference proceedings. ACM Transactions on Computational Logic, March 2001.